

## Introdução à Blockchain e Contratos Inteligentes: Apostila para Iniciante

Lucas Ribeiro (FURG), Odorico Mendizabal (UFSC)

**Relatório Técnico INE 001/2021**



Universidade Federal de Santa Catarina  
Departamento de Informática e Estatística

Copyright © 2019 GSDE (Grupo de Sistemas Disjgitais e Embarcados)

Este trabalho está licenciado sob uma Licença Creative Commons Atribuição-NãoComercial-SemDerivações 4.0 Internacional. Para ver uma cópia desta licença, visite <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

Dúvidas, críticas, sugestões, ou contestações de qualquer natureza, favor encaminhar para os e-mails: [lucasribeiro@furg.br](mailto:lucasribeiro@furg.br) ou [odorico.mendizabal@ufsc.br](mailto:odorico.mendizabal@ufsc.br) com o assunto “**Apostila Blockchain**”.

Versão disponibilizada para os ouvintes do minicurso “Introdução à Blockchain e Contratos Inteligentes com Solidity” realizado na IX Semana Acadêmica do Centro de Ciências Computacionais da Universidade Federal do Rio Grande. Mudanças serão realizadas para publicações futuras, portanto, não altere esse material antes da disponibilização de sua versão final.

*Primeira versão, Agosto de 2019*

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### Conteúdo

Prefácio .....	6
----------------	---

I Blockchain no Mundo Real		
<b>1</b>	<b>Introdução .....</b>	<b>8</b>
<b>1.1</b>	<b>Contexto</b>	<b>8</b>
1.1.1	Evolução do Dinheiro .....	9
	Figura 1.1 .....	9
1.1.2	Gasto Duplo e Centralização .....	9
1.1.3	Conceitos Pré-Blockchain .....	10
1.1.4	Crise do Subprime .....	10
<b>1.2</b>	<b>Proposta</b>	<b>10</b>
	Figura 1.2 .....	11
	Figura 1.3 .....	11
1.2.1	Objetivos .....	11
<b>2</b>	<b>Aplicações em Desenvolvimento .....</b>	<b>13</b>
	Table 2.1 .....	13
<b>2.1</b>	<b>Mercado Financeiro</b>	<b>14</b>
2.1.1	Trading ou Especulação .....	14
	Figura 2.1 .....	14
	Figura 2.2 .....	14
2.1.2	Oferta Inicial de Ativos (Initial Coin Offering, ou, ICO) .....	15
<b>2.2</b>	<b>Tabelionato e Registros</b>	<b>16</b>
	Table 2.2 .....	16
<b>2.3</b>	<b>Indústrias</b>	<b>17</b>
2.3.1	Indústria Energética .....	17

2.3.2	Indústrias Farmacêutica e Alimentícia . . . . .	18
<b>2.4</b>	<b>Governo e Sociedade</b>	<b>18</b>
2.4.1	Dados Públicos . . . . .	18
2.4.2	Registro de Votos . . . . .	18
2.4.3	Transporte Urbano . . . . .	19
2.4.4	Gestão de Saúde . . . . .	19

## II

# Tecnologia por trás do Blockchain

<b>3</b>	<b>Blockchain</b> . . . . .	<b>21</b>
<b>3.1</b>	<b>Cadeia de Blocos</b>	<b>21</b>
	Figura 3.1 . . . . .	22
<b>3.2</b>	<b>Bloco</b>	<b>22</b>
	Figura 3.2 . . . . .	23
<b>3.3</b>	<b>Hash e Endereço</b>	<b>23</b>
<b>3.4</b>	<b>Carteira</b>	<b>23</b>
<b>3.5</b>	<b>Nodos completos</b>	<b>24</b>
3.5.1	Mineração . . . . .	24
<b>3.6</b>	<b>Protocolos de Consenso</b>	<b>24</b>
3.6.1	Prova de Trabalho . . . . .	25
3.6.2	Prova de Participação . . . . .	25
3.6.3	<i>Practical Byzantine Fault Tolerance</i> . . . . .	26
3.6.4	Prova de tempo decorrido . . . . .	26
<b>3.7</b>	<b>Plataformas</b>	<b>27</b>
3.7.1	Ethereum . . . . .	27
3.7.2	EOS . . . . .	27
3.7.3	IBM Hyperledger . . . . .	27
3.7.4	Hyperledger Fabric . . . . .	27
<b>4</b>	<b>Ethereum</b> . . . . .	<b>28</b>
<b>4.1</b>	<b>Especificações</b>	<b>28</b>
	Figura 4.1 . . . . .	28
4.1.1	Ethereum Virtual Machine (EVM) . . . . .	28
4.1.2	Contas . . . . .	29
	Figura 4.2 . . . . .	29
4.1.3	Transações e Mensagens . . . . .	29
4.1.4	Gas . . . . .	30
4.1.5	Ether . . . . .	31
4.1.6	Bloco . . . . .	32
	Figura 4.3 . . . . .	32
4.1.7	Consenso: PoW vs PoS . . . . .	33
4.1.8	Documentação e Ferramentas . . . . .	33
<b>4.2</b>	<b>DApps e Contratos Inteligentes</b>	<b>33</b>

**III****Colocando em Prática**

<b>5</b>	<b>Desenvolvendo Contratos Inteligentes</b>	<b>36</b>
5.1	<b>Mais Contratos Inteligentes</b>	<b>36</b>
5.2	<b>Ambiente de Desenvolvimento: Remix</b>	<b>36</b>
	Figure 5.1 .....	37
5.2.1	Criação de Arquivo .....	37
5.2.2	Compilação e Deploy .....	38
	Figure 5.2 .....	38
5.2.3	Acessando Métodos e Atributos .....	38
5.2.4	Analizando Transações e Chamadas de Funções .....	38
	Figura 5.3 .....	39
	Figura 5.4 .....	39
5.2.5	Contas EOA para Chamada de Transações .....	39
	Figura 5.5 .....	40
5.2.6	Gerando Saídas em JSON .....	40
	Figura 5.6 .....	40
5.3	<b>Solidity</b>	<b>41</b>
5.3.1	"Hello World!" .....	41
5.3.2	Estruturas de Controle de Fluxo .....	42
5.3.3	Herança .....	42
5.3.4	Funções <i>Modifiers</i> .....	43
5.3.5	Unidades e Variáveis Globais .....	43
5.3.6	Estudo de Caso: Conta Bancária .....	44
5.3.7	Vamos Exercitar? .....	45
<b>6</b>	<b>Estudo Guiado</b>	<b>46</b>
6.1	<b>Ethereum Improvement Proposals (EIP)</b>	<b>46</b>
6.2	<b>Criptomoeda e Token</b>	<b>46</b>
6.3	<b>Criando nosso Token</b>	<b>47</b>
6.3.1	ERC-20 .....	47
6.3.2	ERC-721 .....	50
<b>7</b>	<b>Ferramentas Utilizadas</b>	<b>51</b>
7.1	<b>Ferramentas e Notícias</b>	<b>51</b>
7.1.1	Diagramas .....	51
7.1.2	Ícones .....	51
7.1.3	Captura de Código .....	51
7.1.4	LaTeX .....	51
7.2	<b>Notícias</b>	<b>51</b>
	<b>Bibliography</b> .....	<b>53</b>
	<b>Artigos</b>	<b>53</b>
	<b>Livros</b>	<b>54</b>
	<b>Outros</b>	<b>54</b>
	<b>Index</b> .....	<b>55</b>

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### Prefácio

Esta apostila visa apresentar os conceitos elementares da tecnologia Blockchain e do desenvolvimento de Contratos Inteligentes - de maneira simples e inteligível - para alunos de graduação nas áreas de Engenharia, Computação, Administração e afins, além de pesquisadores iniciantes e entusiastas.

O material é dividido em três níveis de atribuições interdependentes: **Blockchain no Mundo Real, Tecnologia por Trás do Blockchain e Colocando em Prática**.

Para os exercícios práticos, é aconselhável que se tenha conhecimentos prévios em lógica computacional, algoritmos e estruturas de dados, além do domínio de alguma linguagem de programação de alto nível (como C/C++, Java, Javascript ou Python), pois as mesmas servem como base para terminologias e sintaxes que serão apresentadas. Contudo, a ausência destas competências não deve servir como limitador no estudo de Blockchain, visto que o trabalho apresenta-se como uma introdução ao tema.

Buscamos apresentar o conteúdo de maneira prática, comparando a tecnologia com situações cotidianas e abstraindo as nomenclaturas rebuscadas, mas sem deixar de lado os conceitos fundamentais. O leitor irá se deparar com uma série de indicações para artigos, livros, e outros tipos de conteúdos que fundamentam toda essa apostila, visto que tentamos trazer uma abordagem informativa para aqueles que se interessam pelo tema. Se possível, não deixe de acessar as leituras indicadas, e também buscar outras fontes de conhecimento.



# Blockchain no Mundo Real

<b>1</b>	<b>Introdução .....</b>	<b>8</b>
1.1	Contexto	
1.2	Proposta	
<b>2</b>	<b>Aplicações em Desenvolvimento .....</b>	<b>13</b>
2.1	Mercado Financeiro	
2.2	Tabellionato e Registros	
2.3	Indústrias	
2.4	Governo e Sociedade	

# APOSTILA PARA INICIAINTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### 1. Introdução

Inúmeras pesquisas, projetos, empresas e startups tem surgido com o propósito de explorar o uso e criação de Blockchains. Entretanto, a grande maioria do material de caráter introdutório sobre o tema, possui grande foco na exploração da tecnologia como uma forma de investimento, dando pouca atenção a suas peculiaridades técnicas e na possibilidade de aplicação dessa tecnologia em outras áreas.

Nessa apostila buscaremos entender a Blockchain a partir de um ponto de vista social e computacional, investigando e relatando de forma simplificada seus protocolos, algoritmos e paradigmas, além de conceituar e explorar **Contratos Inteligentes**, que são trechos de códigos responsáveis por conceber aplicações que executam em Blockchain.

Na primeira parte apresentamos uma contextualização do Bitcoin, seguido por exemplos de indústrias que utilizam de redes Blockchain em alguma parte de seus processos produtivos. Em sequência, na segunda parte, estudamos mais a fundo os conceitos técnicos responsáveis por assegurar as vantagens do uso da tecnologia. E, por fim, na terceira parte, colocamos a mão na massa para a implementação de Contratos Inteligentes, utilizando a linguagem de programação Solidity.

#### 1.1 Contexto

O Bitcoin foi apresentado em 2008 como uma moeda digital revolucionária, tanto no setor financeiro, quanto tecnológico. Em dezembro de 2017, o valor da moeda chegava a aproximadamente U\$20 mil dólares e hoje movimenta bilhões de dólares no sistema financeiro mundial. Parte desse sucesso deve-se a sua infraestrutura descentralizada, pública e imutável, chamada Blockchain.

Blockchain pode ser definido como uma coleção de ferramentas algorítmicas, que, em harmonia, solucionam problemas presentes em transações entre duas partes. O contexto central - necessário para se entender a importância da Blockchain - é a automatização do indivíduo intermediário, responsável pelo julgamento e processamento necessário para que uma **transação** seja considerada justa e segura para ambas as partes. Antes da tecnologia Blockchain, esse papel era realizado majoritariamente pelos bancos e outras instituições presentes na sociedade, como cartórios, agências de regulação, corretoras, etc., porém, veremos que confiar na centralização de tarefas tão sensíveis e importantes pode ser um problema.

### 1.1.1 Evolução do Dinheiro

Desde a antiguidade, a forma monetária como nos relacionamos é um dos grandes pilares do desenvolvimento da sociedade. Inicialmente um agricultor com produção excedente ao seu consumo entraria em contato com um pescador, com semelhante excedente, e então, ambos faziam a troca dessas mercadorias de forma a saciar as necessidades mútuas. Apesar da prática - também conhecida como escambo - ainda ser utilizada em muitas regiões, principalmente em períodos de guerra, ela possui muitas falhas.

O bem de produção de um pescador é composto, majoritariamente, por peixes. Isso significa que um pescador que desejava se alimentar de arroz, estaria submetido a existência de um agricultor responsável pela produção de arroz, que a produção desse agricultor fosse excedente, e que o agricultor possuísse o desejo de consumir peixes. Esse conjunto de requisitos presentes no escambo, levaram a ascensão de produtos que eram de necessidades comuns - como o sal, os tecidos, e posteriormente, os metais - a se tornarem moedas "globais".

Metais como o ouro e a prata possuíam algumas vantagens que facilitaram o desenvolvimento do comércio, como a possibilidade de divisões fracionais, a raridade e a facilidade de transporte. Porém, com o passar o tempo, algumas questões passaram a ser levantadas: as pessoas começaram a perceber que não precisavam mais fisicamente do metal para realizar as trocas comerciais. Além disso, havia o perigo inerente de manter os metais em sua posse, ou de fazer o seu transporte por regiões instáveis. Eis que surgiram os ourives. No período medieval, o ouro era depositado em posse dos ourives, que garantiam a sua segurança. Como forma de manter o controle da posse, os ourives emitiam títulos que representavam a quantia de metal que uma pessoa poderia sacar das oficinas ourives (como um recibo). Desenvolvendo essa prática, essas oficinas ourives consequentemente originaram os primeiros bancos. Assim como os títulos emitidos viriam a se tornar as primeiras moedas em papel com lastro em metal, ou seja, um pedaço de papel que representava a existência garantida de um metal em outro local.

Com a recente globalização da economia, esse padrão de moedas lastreadas (ou Padrão-ouro) entrou em colapso, dando lugar ao que conhecemos como **Moeda Fiduciária**. Estas são moedas que não possuem nenhum valor intrínseco, ou seja, todo o seu valor é resultante da confiança que as pessoas tem no seu emissor (em maioria, os Bancos Centrais). Como consequência, os Estados, Bancos e outras instituições financeiras passaram a ter grande controle na execução de manobras intervencionistas sobre essas moedas. Como exemplo temos a emissão de cédulas, controle das taxas de juros, criação de mercado futuro, entre muitas outras.

Mais a frente, veremos que esse excesso de controle possui falhas e pode ocasionar grandes crises financeiras.<sup>1</sup>



Figura 1.1: Evolução do Dinheiro na Sociedade

**Definição 1.1.1 — Moeda.** Conceitualmente, uma moeda deve apresentar três funções primárias: ser **reserva de valor, unidade de conta e meio de troca**. Devido a grande volatilidade do seu valor, muitos especialistas afirmam que o Bitcoin não pode ser utilizado para as duas últimas, e, por isso não é considerada oficialmente uma moeda. [28]

### 1.1.2 Gasto Duplo e Centralização

Após a ascensão da Internet no início dos anos 2000, houve uma crescente demanda pelo comércio eletrônico em todo o mundo e os pagamentos virtuais surgiram como uma representação das moedas estatais, porém, apresentavam pontos de vulnerabilidades.

O mais comum deles é conhecido como **Gasto Duplo**. Imagine que você acabou de fazer uma compra virtual com os últimos R\$100,00 disponíveis na sua conta bancária, porém, antes que a

<sup>1</sup>Seção escrita com base no vídeo de José Kobori: "O Bitcoin pode se tornar realmente uma moeda no mundo real?"

transação fosse aprovada pelo Banco você fez outra compra utilizando dos mesmos R\$100,00. Como o Banco deveria lidar com essa situação?

Talvez a solução que você tenha pensado foi: “Crie um sistema (ou servidor) que bloquee o seu saldo, valide sempre a primeira transação e cancele as outras!”, correto? Agora imagine que um invasor (pejorativamente conhecido como hacker) decidiu atacar esse servidor, que é o **único** responsável por essa validação. Ou então, se por algum eventual erro de programação cometido por um programador inexperiente, esse servidor “caiu” e ficou fora de funcionamento por dias. Quanto prejuízo isso poderia causar?

Ou, pior ainda, imagine que o governo do seu país iniciou uma revolução de Estado e decidiu que apenas uma parcela da população está autorizada a realizar transações naquele servidor. Apesar de serem situações extremas, elas podem ocorrer em qualquer sistema - umas mais que as outras -, porém, essa probabilidade é muito maior em sistemas centralizados, onde o ponto de falha é único e de acesso mais fácil.

Isso é o que chamamos de **Ponto Crítico de Falha** (ou, SPOF).

### 1.1.3 Conceitos Pré-Blockchain

Ná década de 90, os pesquisadores Stuart Haber e W. Scott Tornetta apresentaram em seus artigos **“How to Time-stamp a Digital Document”** [10] e **“Improving the Efficiency and Reliability of Digital Time-Stamping”**, uma solução computacional prática para autenticação de dados - através de um “carimbo” com data e hora de sua criação - , onde esses dados (em geral, documentos) não podiam ser adulterados. A solução foi baseada em uma lista de blocos encadeados que armazenavam os dados, e, posteriormente, foi complementada por uma estrutura de dados chamada Árvore de Merkle que possibilitava a inclusão de vários documentos dentro de um único bloco.

Entretanto, essa solução ficou em desuso até 2004, ano de término de sua patente. Coincidemente (ou não), no mesmo período, Hal Finney, iniciou um projeto chamado **“RPoW: Reusable Proofs of Work”**, onde apresentou um algoritmo de **consenso por prova de trabalho** que permitia a transferência de ativos (ou moedas), solucionando o problema de gasto duplo. O projeto foi baseado em trabalhos já existentes - como **“Pricing via Processing, Or, Combatting Junk Mail, Advances in Cryptology”** de Cynthia Dwork - , por isso é difícil atribuir um “criador original” à proposta.

### 1.1.4 Crise do Subprime

Durante os muitos séculos de desenvolvimento do capitalismo e das relações monetárias, o mundo vivenciou muitos períodos de crises, sendo a mais atual conhecida como **Crise do Subprime**.

Incentivados pelo governo, alguns dos maiores bancos americanos forneciam empréstimos de forma negligente para famílias economicamente frágeis, com o objetivo de fomentar o setor imobiliário. Títulos de investimentos - denominados *Subprimes* - eram lastreados nessas dívidas e classificados por agências como “investimento seguro”, e passaram a ser vendidos pelos grandes bancos. Porém, após o governo americano forçar um aumento da taxa de juros (como controle da inflação), o preço dos imóveis despencou, e, famílias que estavam em situação de estagnação econômica se viram incapacitadas de refinanciar ou quitar suas dívidas.

Como consequência, um efeito dominó: pessoas se tornaram inadimplentes e os títulos emitidos perderam seu valor, levando investidores e empresas ao prejuízo, por fim, estes se viam obrigados a demitir funcionários ou fechar suas empresas, gerando instabilidade política e econômica na maioria dos países do mundo. Uma semente fora plantada.

Apesar de abstrairmos muito da complexidade da crise, pode-se visualizar o resultado da má interferência das instituições na economia mundial. Dívidas, desemprego, falências e instabilidade abriram precedente para o questionamento do mercado financeiro vigente na época [7].

## 1.2 Proposta

Paralelamente à crise, Satoshi Nakamoto - indivíduo desconhecido, pseudônimo para uma associação ou programador anônimo - publicou o artigo **“Bitcoin: A Peer-to-Peer Electronic Cash System”** [15], que estabeleceu as bases para a criação de uma moeda digital totalmente descentralizada.

A proposta consiste de uma rede ponto-a-ponto (do inglês, peer-to-peer) inspirada nas tecnologias de compartilhamento de música do início do século XXI - como o Napster, Gnutella, Kazaa,



Figura 1.2: Variação do Preço do Bitcoin segundo o Coinbase

Emule, e outros – e fortemente apoiada em conceitos de criptografia e provas de trabalho (como citado anteriormente). A existência dessa rede, viabilizou a criação de uma moeda criptografada (origem do termo **criptomoeda**), livre de interferências e do controle de instituições públicas ou privadas – o Bitcoin.

Em resumo, o artigo reuniu diversas técnicas para solucionar, de forma coerente, os principais problemas enfrentados pelos sistemas de transações centralizados.

Não podemos afirmar que a crise do subprime teve relação direta com o surgimento do Bitcoin, entretanto, com certeza fora um catalisador na estruturação da Blockchain 1.0 – termo utilizado para denominar o uso de redes Blockchain exclusivamente para a criação de criptomoedas – como alternativa de investimentos. Em seguida, a tecnologia se expandiu para outras áreas com a difusão do conceito de **Contratos Inteligentes** (do inglês, Smart Contracts), responsáveis pela origem do termo Blockchain 2.0.

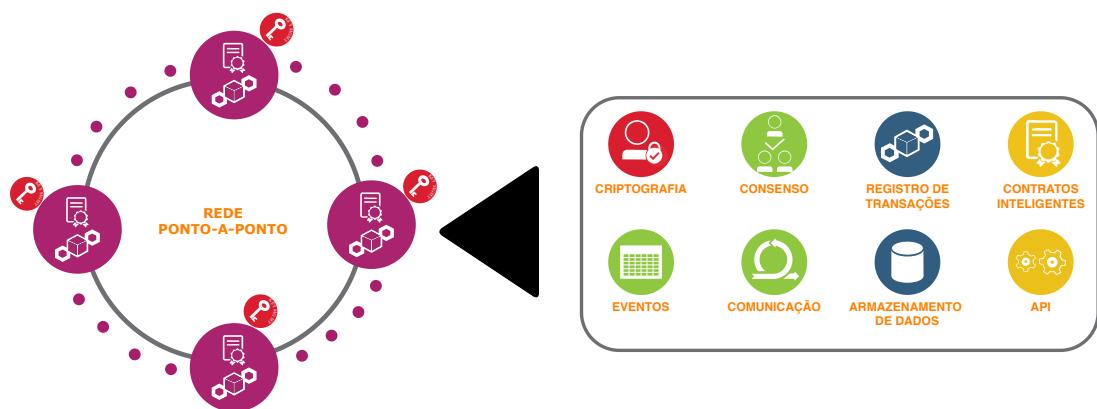


Figura 1.3: Elementos de Uma Rede Blockchain

### 1.2.1 Objetivos

Alguns dos principais objetivos no surgimento do Bitcoin eram:

1. Lidar com os problemas originados por pontos críticos de falha, através da descentralização dos servidores (ou Nós).
2. Lidar com o problema de gasto duplo de maneira eficiente.
3. Garantir segurança e anonimato aos usuários.

4. Globalizar a economia, eliminando as fronteiras geográficas e facilitando as trocas comerciais entre indivíduos de diferentes localidades.
  5. Eliminar” os intermediários com poder de interferência na economia.
- Entretanto, devido a constante evolução das Blockchains, é difícil especificar os limites da sua atuação. No capítulo seguinte veremos algumas das áreas que estão sendo impactadas por essa tecnologia.

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### 2. Aplicações em Desenvolvimento

Muitas são as aplicações que já exploraram das particularidades das Blockchains e muitos são os setores que futuramente poderão ser revolucionados pela sua adoção. Apesar de ser uma tecnologia em fase embrionária, alguns projetos já lideram seus respectivos mercados (tanto em pesquisa quanto em desenvolvimento), logo, nesse capítulo faremos a abordagem dessas aplicações de forma a nortear possíveis estudos.

Atualmente a maioria dos entusiastas categorizam essas aplicações em quatro gerações:

- **Blockchain 1.0:** Criptomoedas e Criptoativos em geral. É a origem das Blockchains, sendo o Bitcoin a primeira e, até hoje, a maior aplicação.
- **Blockchain 2.0:** Contratos Inteligentes, códigos que possibilitam a criação de outros tipos de aplicações além das criptomoedas.
- **Blockchain 3.0:** Aplicações Descentralizadas (ou, DApps). Geralmente possuem um backend de transações executando em uma Blockchain, complementadas por uma interface amigável para interação com usuários em massa. Aqui também se inclui a “evolução” da Blockchain, denominada **Directed Acyclic Graphs** (ou, DAG), um novo algoritmo de encadeamento de transações.
- **Blockchain 4.0:** É a expansão da geração 3.0 para a “Indústria 4.0”. Em suma maioria, são aplicações híbridas, onde há uma rede Blockchain como um serviço responsável pelo registro de transações de uma empresa, complementando todo um sistema automatizado.

Apesar de abordar elementos de todas as gerações, devido ao caráter dessa apostila, focaremos na 1.0 e 2.0, visto que as gerações seguintes são expansões dos seus conceitos.

Geração	Usos	Projetos
1.0	Criptomoedas e Criptoativos	Bitcoin, XRP, Ethereum
2.0	Contratos Inteligentes	Ethereum, EOS, IBM Hyperledger
3.0	DApps, Tecnologia DAG	Fantom, Cardano, IOTA
4.0	Automação, Cloud, Cidades Inteligentes	Multiversum, Metahash

Tabela 2.1: Evolução das Blockchains e Exemplos de Aplicações

## 2.1 Mercado Financeiro

Segundo a Comissão de Valores Mobiliários (CVM), os criptoativos são “ativos digitais, protegidos por criptografia, presentes exclusivamente em registros digitais, cujas operações são executadas e armazenadas em uma rede de computadores”. Originalmente, a intenção da criação desses ativos era facilitar as transações digitais, otimizando o e-commerce e a internet como um todo. Entretanto, devido a sua **alta volatilidade**, hoje as criptomoedas exercem outras funções que são muito exploradas.

**Observação 2.1.1 — Alerta.** Assim como todo o material, esse capítulo possui caráter introdutório, com o objetivo de servir como orientador de estudos, portanto, qualquer prejuízo financeiro resultado do uso equivocado de informações apresentadas aqui, não é de responsabilidade dos autores. Tenha em mente que são muitos os riscos em investimentos de criptoativos, como: fraudes, baixa liquidez, alta volatilidade e desregulamentação. Tenha cuidado.

### 2.1.1 Trading ou Especulação

**Trading** é o nome dado a técnica de analisar o mercado com o objetivo de obter lucro, através de compra e venda baseadas em predição. O valor desses ativos variam bruscamente, logo, é comum uma moeda que ontem valia R\$500,00, amanhã passar a valer R\$1.000,00, sem um motivo aparente ou uma explicação lógica muito profunda. A seguir, temos o exemplo de uma situação inusitada que ocorreu com a criptomoeda "Dogecoin". Após um tweet de elogio feito pelo empresário Elon Musk, a moeda apresentou um aumento de aproximadamente 40%.

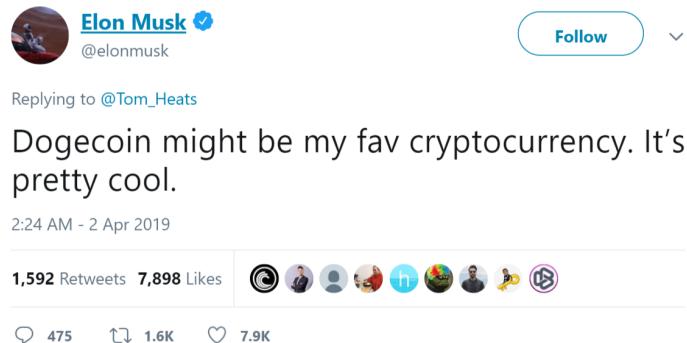


Figura 2.1: Tweet de Elon Musk Sobre a Dogecoin: “Dogecoin talvez seja minha criptomoeda favorita. É bem legal.”



Figura 2.2: Aumento de 40% do Valor da Dogecoin após Tweet. Relatado pelo site Coinmarketcap

Essa alta volatilidade, abre um universo de possibilidade para investidores experientes e

especuladores do mercado financeiro. São três as movimentações mais comuns feitas com criptomoedas, mas que também são herdadas do mercado de ações tradicional:

- **Day Trade:** Baseada em múltiplas trocas diárias, obtendo pequenos lucros em prazo muito curto. É muito comum a utilização de *Bots* que automatizam essa técnica.
- **Swing Trade:** Realiza trocas em períodos de 3 à 7 dias, com lucros intermediários em curto prazo.
- **Buy and Hold:** Consiste em comprar ativos, mantendo-os em sua posse por tempo indeterminado, aguardando por uma grande valorização ao longo prazo. É a técnica favorita de grandes investidores de Wall Street.

Predição é a maneira como um investidor é influenciado a tomar uma posição no mercado (como realizar uma compra ou uma venda), e, apesar de haver uma infinidade de técnicas para se chegar a um resultado ou uma indicação, podemos separá-las informalmente em duas correntes:

- **Análise Financeira e Fundamentalista:** É uma análise qualitativa das instituições que são responsáveis pelos ativos, ou, uma análise do possível desempenho da tecnologia no futuro. São voltadas para o longo prazo e, também, são muito utilizadas no mercado de ações.
- **Análise Técnica ou Gráfica:** Consiste na análise do preço dos ativos em relação ao tempo, fazendo comparações com outros ativos e buscando padrões gráficos que justifiquem uma movimentação de curto prazo no mercado.

### 2.1.2 Oferta Inicial de Ativos (Initial Coin Offering, ou, ICO)

Imagine que você é o proprietário de uma grande empresa de tecnologia, e, apesar da sua empresa estar vivendo um bom momento financeiro, ela precisa de investimentos para expandir sua produção, de modo a fazer frente a seus grandes concorrentes. No mercado de ações tradicional, uma forma de conseguir esses investimentos seria através de uma **Oferta Pública Inicial** (do inglês, Initial Public Offering, ou IPO)

Popularmente conhecido como **abertura de capital**, esse é um evento onde uma empresa emite ativos que representam uma parcela da empresa, e em seguida, os vende em uma bolsa de valores. Normalmente, devido a grande regulamentação do mercado, esse processo seria demorado e muito custoso, porém, com a utilização de redes Blockchains se tornou viável a criação de **tokens**, que são ativos virtuais que representam participação nos lucros ou serviços da empresa. Por ser tratar de um ativo virtual representado por uma criptomoeda, a prática foi batizada como **Oferta Inicial de Ativos** e tem se tornado muito utilizada por Startups.

Funciona da seguinte forma:

1. Em uma rede Blockchain pública, a empresa cria um token, implementado a partir de um Contrato Inteligente;
2. A empresa anuncia a abertura da ICO para o público interessado, junto de uma documentação detalhada de todo o projeto (incluindo um *white paper* e um *road map*);
3. Investidores interessados fazem a aquisição dos tokens. Geralmente o pagamento é feito com Bitcoin, Ethereum ou Dólar Americano;
  - a. Alcançando o valor desejado e condições pré-estabelecidas, a empresa obtém o direito de usar do valor arrecadado para suas operações
  - b. Caso as condições não sejam alcançadas, todo o montante arrecadado é devolvido aos investidores
4. Com o sucesso do projeto, o valor desse token tende a valorizar em grandes proporções, possibilitando ao investidor fazer a revenda por um preço maior que o investido. Caso contrário, o prejuízo terá a mesma proporção;

**Observação 2.1.2 — Alerta.** Note que o investimento em uma ICO é muito arriscado, busque sempre o máximo de informações sobre a empresa ou Startup antes de aplicar seu dinheiro nesse tipo de aplicação. Em grupos e blogs dos maiores entusiastas da área, a recomendação mais citada é que seja feita uma análise das pessoas envolvidas na empresa, como, a reputação dos diretores e o nível técnico dos funcionários.

## 2.2 Tabelionato e Registros

Muitos definem Blockchain como um livro-razão público imutável e descentralizado. Segundo o Wikipedia, **imutabilidade** é uma condição onde um objeto ou dado não pode ser alterado após a sua criação. Mas o que isso significa aqui?

No capítulo anterior, citamos que uma Blockchain é formada a partir de um rede ponto-a-ponto, o que significa que há computadores ao redor de todo o mundo executando a computação necessária para que tudo ocorra bem. A novidade, é que em cada um desses computadores (denominados Nós), existe um cópia de todas as transações já realizadas por essa rede. Isso significa que um depósito de Bitcoins feito por uma pessoa X para outra pessoa Y em 2010, ainda está “salvo” em milhares de máquinas, sem que seja possível alterar qualquer informação dessa transação. Além disso, para que uma nova transação seja inserida nessa rede, é necessário que um Nô responsáveis (denominado Minerador) deve verificar se essa transação é “verdadeira”, e em caso positivo, a maior parte ( $50\% + 1$ ) desses computadores precisam “aceitar” a sua inclusão.

Fica claro, que, um indivíduo mal intencionado precisaria possuir uma capacidade computacional imensamente grande de forma a corromper todas essas máquinas. Dessa forma, foi identificado a possibilidade de utilização dessa tecnologia para outras aplicações que exigem da imutabilidade de dados em um nível crítico, como por exemplo, a autenticidade de documentos públicos e privados usados no dia-a-dia, serviço que hoje é garantido pelos Cartórios e Registros.

Serviço	Exemplos de Documentos
Tabelionato de Notas	Escrituras, Procurações e outros.
Registro de Contratos Marítimos	Transações de Embarcações Marítimas
Protesto de Títulos	Inadimplência e Descumprimento da Obrigaçāo
Registro de Imóveis	Situação e Titularidade de Imóveis
Títulos de Pessoa Jurídica	Registro de Entidades e outros
Registro Civil das Pessoas Naturais	Nascimentos, Casamentos, Óbitos e outros
Registro de Distribuição	Centro de Informações Judiciais

Tabela 2.2: Serviços de um Cartório Segundo a Legislação Brasileira

Por se tratar de um tema delicado, que envolve a incerteza da existência de profissões no futuro e a verdadeira segurança garantida pelas Blockchains, muitos embates estão sendo travados entre profissionais da área dos registros físicos, especialistas da tecnologia Blockchain e os governos. Os mais extremistas acreditam que os Cartórios serão extintos em alguns anos, e por outro lado, muitos exploram as possíveis fragilidades da tecnologia e aspiram seu fracasso. Contudo, o curso mais provável que seja tomado, é o da existência de Cartórios físicos que garantem a segurança dos dados através do uso de redes Blockchain. Situação que pode gerar benefícios mútuos.

Nesse mercado, podemos citar alguns projetos que estão em evidência: no Brasil temos a **OriginalMy**, já mundialmente a **STAMPD**, **Blockusign** e **Blocknotary** são projetos líderes. Para os interessados no segmento, recomendamos a leitura de seus respectivos *white papers* e estudos mais aprofundados no tema.

Projeto	Website
OriginalMy	<a href="http://originalmy.com">originalmy.com</a>
STAMPD	<a href="http://stampd.io">stampd.io</a>
Blockusign	<a href="http://blockusign.co">blockusign.co</a>
Blocknotary	<a href="http://blocknotary.com">blocknotary.com</a>
OriginStamp	<a href="http://originstamp.org/home">originstamp.org/home</a>

Tabela 2.3: Projetos que Exploram os Registros de Documentos por Blockchain

Nos capítulos subsequentes explicamos mais sobre os algoritmos responsáveis por garantir a persistência e imutabilidade dos dados gravados em uma rede Blockchain.

## 2.3 Indústrias

Com a crescente necessidade de automação das indústrias atuais (fenômeno conhecido como indústria 4.0), viu-se que as Blockchains poderiam ser adaptadas para a solução de problemas recorrentes em alguns setores produtivos.

### 2.3.1 Indústria Energética

Quando consumimos energia elétrica em nossas casas somos cobrados pelo consumo em kWh. Não sabemos se aquela energia foi originada por uma usina hidrelétrica, termelétrica ou alguma fonte renovável, pois toda a energia será transformada para uma única unidade de medida. Assim, identificou-se que o mercado de recursos energéticos poderia explorar do potencial das redes Blockchain para a resolução de desafios enfrentados nesse comércio.

No artigo “**Blockchain technology in the energy sector: A systematic review of challenges and opportunities**” [1], os autores apresentam algumas possibilidades de uso das Blockchains nesse contexto, onde se destacam:

- **Gerenciamento das Redes (do inglês, Grid):** Auxiliar no gerenciamento das redes descentralizadas, flexibilização de serviços e gerenciamento de ativos. Por outro lado, também pode afetar o custo da energia.
- **Gerenciamento de Identidades e Segurança:** Utilizar dos algoritmos de criptografia propostos pelas redes Blockchain para proteger a privacidade das partes envolvidas.
- **Compartilhamento de Recursos:** Forma de cobrar o compartilhamento de recursos entre vários usuários, como infraestrutura de carregamento ou armazenamento de dados.
- **Concorrência:** Contratos Inteligentes poderiam acelerar a “troca” por fornecedores de energia, o que aumentaria a concorrência entre as companhias e reduziria o valor das tarifas.
- **Transparéncia:** Registros, transações e processos armazenados de forma persistente e transparente, o que facilitaria as auditorias e regulações.

Outra forma de aplicar Blockchain na indústria energética, seria na negociação de *Créditos de Carbono*, resultantes do uso de energia renovável. Estabelecido após uma série de acordos internacionais que visam combater as mudanças climáticas no planeta Terra, esses créditos representam unidades de Dióxido de Carbono que uma entidade pode emitir em uma atividade produtiva.

Imagine que você mora em um país desenvolvido e industrializado – como o Reino Unido ou os EUA –, e deseja abrir uma fábrica que irá emitir muitos gases poluentes. Como esses países já atingiram a quantidade “recomendada” de emissão, provavelmente você terá problemas burocráticos que irão lhe impedir de abrir essa fábrica. Uma solução para esse problema seria a aquisição de créditos de carbono gerados por outros países que emitem abaixo do seu limite recomendado ou que possuem uma matriz de energia renovável reconhecida por órgãos internacionais.

Assim como toda transação financeira, essa aquisição de créditos possui muitos possíveis pontos de falhas, e, talvez por isso, não é tão difundida no mercado. Logo, empresas e Startups surgiram com a proposta de utilizar redes Blockchain para automatizar esse processo, utilizando de suas características para resolver muitos dos problemas.

Na Tabela 2.4, vemos alguns projetos que já utilizam desses conceitos. Além dessas, há também no Brasil algumas empresas que seguem a mesma linha, como: **GREENCHAIN**, **Enercred** e **FOHAT**.

Projeto	Atividades	Plataforma	Região
Alliander	Troca de energia descentralizada	Ethereum	Países Baixos
Assetron Energy	Criptoativos e Investimentos	Waves	Austrália
CarbonX	Certificados e Créditos de Carbono	Ethereum	Canadá
Tennet & Sonnen	Gerenciamento de Redes (Grid)	Hyperledger Fabric	Países Baixos

Tabela 2.4: Projetos que Exploram Blockchain na Indústria Energética

### 2.3.2 Indústrias Farmacêutica e Alimentícia

Nesses mercados, as Blockchains se apresentaram como solução para um problema de décadas: a dificuldade de obtenção de dados **transparentes** na gestão da cadeia de suprimentos. Antes de chegar à nossa mesa, um bem de consumo, passa por uma série de procedimentos (logísticos, químicos, etc.) e muitas vezes – devido a ausência de informações – se torna difícil para o gestor de uma empresa manter o controle de qualidade sobre o produto em todos esses processos.

As consequências dessa falta de informação são variadas, como: contaminação de produtos, fraudes dos intermediários e baixa predizibilidade de preços e demandas. Assim, por se tratar de um banco de dados descentralizado, as redes Blockchain permitem que o acesso aos dados seja feito de maneira mais eficiente. Mitigando esses efeitos, as empresas podem melhorar os seus serviços aos consumidores, ao mesmo tempo que aumentam seus lucros.

Alguns exemplos de empresas que utilizam ou estão desenvolvendo soluções em Blockchain para sua cadeia produtiva:

- Alimentício: Walmart, Carrefour, Nestlé e Unilever.
- Farmacêutico: McKesson Corporation, AmerisourceBergen Corporation, Premier Inc. e Pfizer Inc.

O projeto pioneiro dessas aplicações é conhecido como **IBM Food Trust**, onde a empresa de tecnologia IBM em parceria com gigantes do varejo, busca soluções integradas com IoT e Blockchain para o rastreamento de produtos em tempo real.

**Observação 2.3.1 — Nota.** É comum dizer que todo dado presente em uma Blockchain é **público**, porém, esse termo só é aplicado para Blockchains públicas como o Ethereum. Em Blockchains permissionadas, onde o acesso é garantido apenas para um grupo de pessoas, esses dados podem ser restritos.

## 2.4 Governo e Sociedade

Devido às propriedades que promovem **segurança, persistência e transparência** de dados, muitos países ao redor do mundo estão adotando redes Blockchain em sua infraestrutura governamental. Alguns exemplos de países que já anunciaram medidas são: Estônia, Singapura, China, Emirados Árabes (Dubai), Japão e Canadá.

### 2.4.1 Dados Públicos

Intuitivamente, a primeira aplicação que podem explorar dessas redes, é no registro e armazenamento de dados públicos. No Brasil, é comum vermos notícias envolvendo fraudes em documentos sensíveis como CPF, carteira de habilitação e registros de posse de terras. Salvos em uma Blockchain, essas fraudes poderiam ser reduzidas consideravelmente ou até mesmo extintas, pois, como citado anteriormente, a possibilidade de alteração de um dado já salvo em uma Blockchain exige uma capacidade computacional muito alta, o que hoje só pode ser alcançada por computadores quânticos.

A transparência também pode ser aplicada para dados auditáveis, como os gastos do governo, assim, todo cidadão poderia acessá-los, e, dessa forma podendo reduzir a corrupção institucional.

### 2.4.2 Registro de Votos

Em 2018 foi relatado pelo Instituto Internacional para a Democracia e a Assistência Eleitoral que além do Brasil, o sistema de *Urna Eletrônica* é utilizado em mais de 30 países, entretanto, há muito debate a respeito da segurança desses sistemas.

Uma equipe de segurança da informação da UNICAMP realizou, em 2017, uma série de testes nas urnas utilizadas nas eleições brasileiras, e o resultado foi a descoberta de inúmeras falhas no sistema. A equipe foi capaz de alterar a mensagem mostrada para o usuário após o registro de seu voto e também teve progresso nas tentativas de desvio de votos.

Esse é apenas um dos relatos que comprovam a fragilidade da tecnologia utilizada atualmente e por isso a difusão de sistemas de votos eletrônicos ainda sofre resistência de muitas organizações. Alguns projetos buscam fornecer um sistema de votos mais seguro do que os sistemas tradicionais,

utilizando das tecnologias utilizadas nas redes Blockchain através de Contratos Inteligentes. Como exemplos temos: **Agora** e **WaveVote**.

#### 2.4.3 Transporte Urbano

A Secretaria Municipal de Planejamento de Teresina (Piauí) anunciou em 2018 uma parceria com IBM Hyperledger para a implementação de uma rede Blockchain para auxiliar na mobilidade urbana da cidade. A ideia é que o sistema seja utilizado para armazenamento de informações relativas ao transporte coletivo, como: cumprimento de ordens de serviço, relatórios de viagem e outros.

#### 2.4.4 Gestão de Saúde

Sistemas de gestão hospitalares muitas vezes são confusos e mal aplicados. É comum que um paciente precise refazer seu cadastro com informações básicas em diferentes unidades hospitalares de uma mesma região.

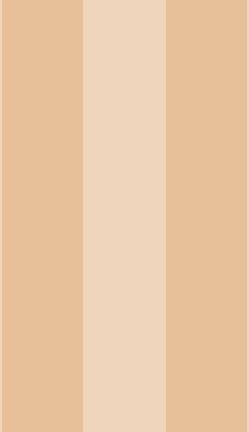
Com o uso de uma rede Blockchain, todo o histórico de consultas, exames e receitas indicadas a um paciente pode ser armazenada de forma persistente e de acesso transparente por qualquer unidade de saúde. Como consequências podemos citar:

- Maior eficiência no serviço com a maior agilidade no processamento de dados dos pacientes.
- Redução de erros humanos provocados por dados incorretos ou inconsistentes.
- Evolução dos serviços, onde, por exemplo, laboratórios podem ter acesso a informações consistentes que ajudem na otimização das técnicas.
- Maior controle sobre armazenamento e logística de medicamentos. Entre outras.

Dos projetos pioneiros no desenvolvimento de tecnologias médicas com uso de Blockchain, temos: **MedRec**, **MedicalChain**, **Pokitdok** e **GUARDTIME**, que são projetos apoiados por grandes instituições de pesquisa e desenvolvimento de tecnologia como a Microsoft, o MIT e a IBM.

**Observação 2.4.1 — Curiosidade.** A Estônia é, provavelmente, o país com maior foco em investimento e desenvolvimento de tecnologias em Blockchain e IoT no mundo. O país está em fase de implementação do projeto **e-Estonia** que visa digitalizar e automatizar todos os serviços públicos.

Além dessas, muitas outras aplicações para Blockchain estão sendo exploradas no Mundo, e para acompanhar essas descobertas o leitor pode utilizar das seguintes fontes: <https://stateofthedapps.com> e <https://www.upfolio.com/collections> que possuem portfólios completos dessas aplicações, além de conteúdos interessantes.



# Tecnologia por trás do Blockchain

<b>3</b>	<b>Blockchain</b>	<b>21</b>
3.1	Cadeia de Blocos	
3.2	Bloco	
3.3	Hash e Endereço	
3.4	Carteira	
3.5	Nodos completos	
3.6	Protocolos de Consenso	
3.7	Plataformas	
<b>4</b>	<b>Ethereum</b>	<b>28</b>
4.1	Especificações	
4.2	DApps e Contratos Inteligentes	

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### 3. Blockchain

Como já vimos, as Blockchains foram desenvolvidas como plataformas para criptomoedas e, como qualquer moeda, foi necessário implantar mecanismos de segurança, prevenindo a falsificação dos ativos, além de garantir autenticidade dos proprietários dos recursos. Entidades financeiras são os responsáveis por garantir tal segurança em transações com moedas físicas. Porém, em um sistema descentralizado, como a Blockchain, protocolos e tecnologias específicas são empregados para garantirem segurança, disponibilidade, além das propriedades de imutabilidade da cadeia de blocos. Este capítulo apresenta os principais conceitos e tecnologias por trás de uma Blockchain.

#### 3.1 Cadeia de Blocos

Uma cadeia de blocos (Blockchain<sup>1</sup>), inicialmente utilizada pelos sistemas de moedas virtuais, pode ser descrita como um arquivo digital, que se mantém atualizado e disponível a todos os usuários da rede. Esta cadeia pode ser vista como um livro razão, contendo todas as transações já realizadas e verificadas, em ordem cronológica. Livro-razão é um livro do processo contábil por meio do qual é possível controlar separadamente a movimentação de todas as contas de uma empresa.

Uma transação inserida na cadeia de blocos ficará lá permanentemente, servindo como prova de sua validação. No caso do Bitcoin, por exemplo, esse livro-razão serve para o registro de transações que utilizam a moeda. Porém, em uma forma mais ampla, este livro serve para o registro de qualquer ativo desejado por um usuário, como registro de nascimento, certidão de casamento, registros de imóveis, recibo de serviço realizado ou qualquer outro documento [20].

A Figura 3.1 ilustra uma transação sendo adicionada à cadeia de blocos. Alice quer enviar a quantia para Bob através de uma transação. Então a transação de Alice é adicionada à um bloco, juntamente com outras transações. Este bloco é disseminado pela rede e, através de um protocolo de consenso, participantes certificam que o bloco é válido e decidem adicioná-lo na cadeia de blocos. Uma vez na cadeia, o bloco que contém a transação de Alice passa a integrar o sistema, de forma imutável. A partir deste momento, a transação de Alice tem efeito, transferindo uma quantidade de dinheiro para Bob.

Com relação à infraestrutura onde a cadeia de blocos é implantada e permissão de acesso, a cadeia de blocos pode ser dividida em três tipos:

1. **Privada:** Apresenta um ambiente controlado, com um número reduzido de nodos, maior controle sobre a identidade dos participantes e maior nível de centralização. Esse tipo de

<sup>1</sup>Os termos cadeia de blocos e *Blockchain* são usados como sinônimo nesta apostila.

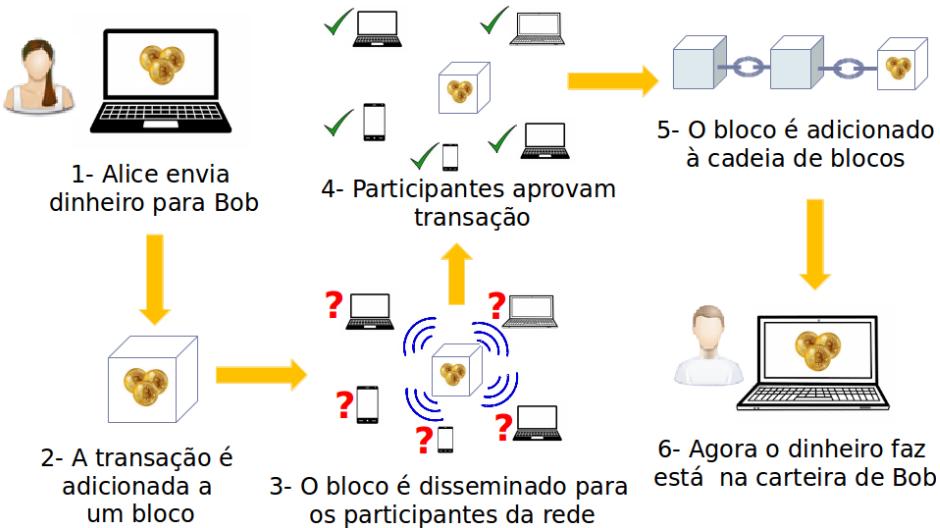


Figura 3.1: Ilustração de uma transação sendo inserida na cadeia de blocos.

sistema apresenta vantagens, por exemplo, para uso em redes empresariais, onde é preciso identificar os funcionários que participam da rede, de acordo com suas permissões. Esta modalidade de redes privadas, lançadas para grandes empresas, também são chamadas de *cadeias de blocos permissionadas*.

2. **Pública:** Rede aberta a qualquer usuário, sem identificação dos nodos e totalmente descentralizada. Aplicável na maioria dos sistemas de moedas virtuais, pois funciona bem para um grande número de nodos, em ambientes onde não é necessário conhecer a identidade real dos participantes.
3. **Consórcio:** Misto entre público e privado. Pode apresentar acesso público e aberto, mas mantendo algum nível de centralização, onde um número menor de nodos pode possuir maiores privilégios para controlar o fluxo de transações na rede.

### 3.2 Bloco

Bloco é um arquivo que contém o registro de diversas transações em espera para serem anexadas na cadeia de blocos. É possível comparar cada bloco como uma das folhas de um livro-contábil. Os blocos são sempre adicionados em ordem cronológica ao fim da cadeia de blocos, de onde nunca poderão ser removidos após sua validação. Cada bloco, para ser validado, precisa incluir certas informações, como a resposta correta do desafio criptográfico, o tamanho do bloco, o cabeçalho, os registros das transações contidas no bloco e a referência ao bloco imediatamente anterior [2, 17]. Como a cadeia de blocos está sempre disponível a todos os usuários da rede, é simples consultar o registro de um bloco específico.

A Figura 3.2 ilustra um bloco do sistema Bitcoin. Sobre os campos:

- **Version** indica a versão do sistema utilizada pelo usuário que propôs o bloco.
- **Previous block hash** é a *hash* invertida do bloco imediatamente anterior.
- **Merkle root** é a *hash* invertida que indica a nodo raiz de uma estrutura chamada *Merkle tree*, utilizada para organizar os blocos da cadeia. Esta estrutura é utilizada em sistemas Blockchain pois possibilita uma busca eficiente por transações contidas na cadeia.
- **Timestamp** é a marcação de hora e data de criação do bloco, baseado em horário UNIX<sup>2</sup>. Um *timestamp* é considerado válido se for maior do que a média de *timestamps* dos 11 blocos anteriores. Este campo também pode ser usado para dificultar a manipulação maliciosa da cadeia de blocos [15].
- **Bits** representa a dificuldade base do sistema no momento em que o bloco foi proposto.
- **Nonce** é um número único utilizado para a resolução do desafio criptográfico.

<sup>2</sup>Horário UNIX corresponde a contagem de segundos desde o dia 1º de janeiro de 1970, no horário 00:00:00 UTC.

- **Transaction count** informa a quantidade de transações no bloco.
- **Coinbase transaction** é a primeira transação do bloco. Ela indica quem vai receber a recompensa do desafio criptográfico.
- **Block hash** é a *hash* gerada a partir de todas os dados do bloco.

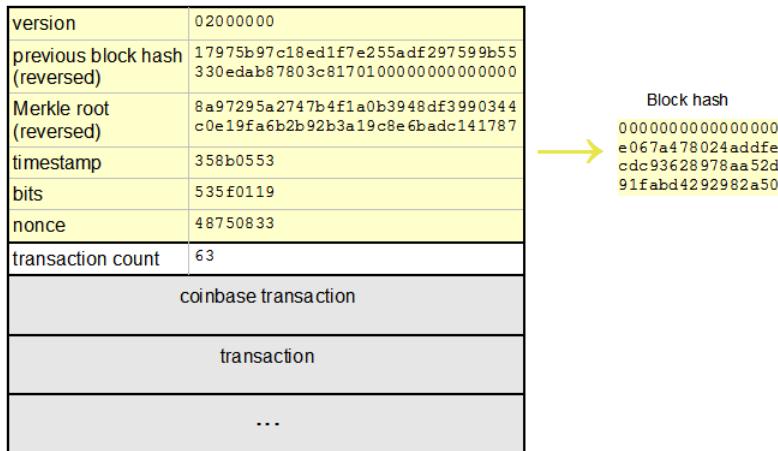


Figura 3.2: Representação simplificada de um bloco Bitcoin. Fonte: Ken Shirriff [22].

### 3.3 Hash e Endereço

Hash é uma função matemática responsável por transformar uma quantidade de **dados** de tamanho indefinido, em uma cadeia **única** de caracteres de tamanho fixo. Imagine que você possui um livro com 500 páginas, e deseja encontrar uma “frase” que resuma todo aquele livro. Para fazer isso, basta colocar todo o conteúdo daquele livro em uma função **Hash** (como o SHA-256 ou o MD5) que você terá um resumo feito com caracteres aleatórios e (praticamente) indecifráveis.

Se tentar alterar apenas 1 caractere desse livro de 500 páginas, toda a cadeia final da Hash será diferente da Hash anterior, e esta é uma das propriedades mais interessantes das funções Hash. Assim, foi criado o “desafio” de uma Blockchain: um minerador insere caracteres no conjunto de transações que ele deseja formar em um bloco, até que a Hash final possua uma quantidade definida de zeros no início da sua cadeia. E para encontrar esse conjunto de caracteres, só há uma opção: inserção aleatória (força bruta).

Já um **Endereço**, é basicamente o Hash de uma chave pública. Seu papel é servir como uma segunda “camada” de segurança para os usuários, e também auxiliar na verificação de uma transação. Sempre que ler o termo **endereço** nessa apostila, pode-se abstraí-lo como sendo o número de uma conta bancária.

### 3.4 Carteira

Para ingressar em qualquer sistema Blockchain, o usuário precisa instalar um aplicativo, também chamado de carteira, que permite que ele realize transações. A carteira contém a chave privada do usuário, que será utilizada para assinar suas transações. Processos responsáveis por validar as transações deverão, então, decifrar as transações utilizando a chave pública do usuário, de modo a validar a transação. Este procedimento, denominado criptografia assimétrica [23] confere ao sistema segurança, permitindo verificar a autenticidade dos autores das transações.

Há diversos tipos de carteiras disponíveis, todas com suas vantagens e desvantagens, sendo um dos principais pontos a se considerar a segurança do usuário, bem como seu anonimato, além da ausência de algum ente centralizador controlando o sistema de carteira. Por exemplo, a página do Bitcoin<sup>3</sup> elenca cinco atributos para escolha da carteira:

<sup>3</sup>Bitcoin: <http://www.bitcoin.org>

1. **Controle sobre seu dinheiro:** se positivo, significa que nenhum ente terceiro pode congelar ou perder os fundos do usuário, que, porém, é responsável pela segurança e constante *backup* de sua carteira;
2. **Validação:** se a carteira é ou não um nodo completo, ou seja, capaz de realizar a verificação e a validação de transações na rede, sem a necessidade de um ente terceiro no processamento;
3. **Transparência total:** significa que a carteira possui código aberto, tornando viável que qualquer usuário audite o código e verifique se não há funções maliciosas inseridas;
4. **Ambiente vulnerável:** se a carteira pode ser instalada em um ambiente vulnerável a *malwares*. Recomenda-se, nesses casos, utilizar senhas fortes, armazenamento offline e autenticação em duas etapas.
5. **Privacidade otimizada:** relaciona-se com a segurança que a carteira oferece, roteando endereços e tornando difícil o rastreamento e espionagem do usuário. Além disso, uma carteira segura não emite informações desnecessárias na rede e permite o uso de serviços online de *anonimização*.

### 3.5 Nodos completos

Um nodo completo é uma entidade da rede Blockchain que executa um programa capaz de validar totalmente as transações realizadas, verificando todas as transações e blocos que recebe e então os transmite a outros nodos completos na rede, para a inserção na cadeia de blocos. Os nodos completos também realizam verificação e validação de transações realizadas por usuários comuns, que apenas utilizam o sistema para transações, como por exemplo, transações financeiras em sistemas de criptomoedas.

Um usuário que deseja se tornar um nodo completo precisa investir tempo e dinheiro, pois o esforço computacional necessário é grande. Além da utilização constante de dados de rede, este precisa, também, tomar diversas precauções de segurança para manter a rede o mais distante possível de ataques que ele próprio possa sofrer. Outro grande problema que esses usuários podem enfrentar são restrições legais, visto que, em alguns lugares, atividades envolvendo Blockchains, especialmente criptomoedas, como o Bitcoin, são proibidas [2].

#### 3.5.1 Mineração

O trabalho de mineração consiste em realizar esforço computacional para a descoberta de novos blocos e sua inserção na cadeia de blocos. Diferentes protocolos de consenso podem ser utilizadas para a descoberta que qual bloco deve ser inserido na rede, conforme detalhamos na Seção 3.6. Em cadeias de bloco públicas, um dos protocolos de consenso mais utilizados é o protocolo de prova-de-trabalho. Devido ao grande esforço computacional necessário para minerar na rede, ou seja, determinar qual bloco deve ser inserido na cadeia, infraestruturas computacionais de alto desempenho especializadas acabam sendo utilizadas [9]. No contexto de Bitcoin, estas muitas vezes são chamadas de minas de bitcoins, que são *clusters* de mineradoras ativas.

#### Minerador

Em resumo, um minerador é um nodo responsável por realizar o trabalho de mineração, que como já vimos, consiste em: (i) validar um conjunto de transações, (ii) formar um bloco com esse conjunto de transações, (iii) encontrar o **Hash** e o **Nonce** de um bloco (resposta do desafio), além de (iv) disseminar esse bloco na rede, que pode ou não ser aceito.

### 3.6 Protocolos de Consenso

Protocolos de consenso são amplamente utilizados em sistemas distribuídos, permitindo que um conjunto de processos independentes concorde sobre um mesmo valor proposto [25]. Estes protocolos estão presentes em implementações de sistemas replicados, podendo também ser adotados para o desenvolvimento de protocolos de comunicação que ofereçam ordenação total entre mensagens trocadas, sendo este um bloco básico de construção de sistemas tolerantes a faltas, comunicação em grupo, entre outras aplicações.

Sistemas baseados em cadeias de blocos também utilizam protocolos de consenso, com o objetivo de garantir uma decisão descentralizada e uniforme sobre qual bloco deve ser adicionado à

cadeia de blocos [4, 6, 21]. Para esta tomada de decisão, diferentes estratégias podem ser adotadas, sendo as principais abordagens:

1. Qualquer nodo pode propor blocos, e o protocolo verifica qual bloco, entre vários propostos, deve ser incluído na cadeia de blocos.
2. O protocolo define previamente qual nodo tem o direito de propor um bloco na rede.

Uma característica dos protocolos de consenso tradicionais, é que uma pequeno conjunto de processos confiáveis pode decidir sobre um valor proposto. Em sistemas baseados em Blockchains públicas, a decisão pela escolha do bloco não pode ser resolvida por um pequeno conjunto de participantes, visto que isto poderia implicar em uma fragilidade do sistema. Desta forma, protocolos de consenso mais escaláveis são necessários.

### 3.6.1 Prova de Trabalho

Prova-de-trabalho (PoW, do inglês *proof-of-work*) foi adotado pela rede do Bitcoin. Neste protocolo, os mineradores competem entre si para a produção de novos blocos, podendo assim coletar a recompensa e as taxas de transação relativas ao novo bloco descoberto. Esse processo ocorre através da resolução de um desafio computacional, cuja dificuldade é ajustada automaticamente para que a taxa média de produção de novos blocos se mantenha um novo bloco a cada 10 minutos, em média [15]. Este desafio consiste em encontrar um valor para o **Nonce** que produza uma **Hash** para o bloco iniciando com um determinado número de bits em 0. Esse **Nonce** é um campo de 32 bits que pode assumir qualquer valor, fazendo com que sejam necessárias muitas tentativas até que o número  $n$  de bits 0 seja alcançado, garantindo que grande esforço computacional foi empregado na solução do problema [12], daí o nome Prova de Trabalho.

Caso mais de um nodo consiga resolver o desafio criptográfico e propor um bloco para uma mesma posição da cadeia, a decisão majoritária é representada pela cadeia mais longa, que possui o maior esforço computacional investido. Se um número de blocos está sendo produzido muito rapidamente, a dificuldade da prova-de-trabalho aumenta, assim como ela pode diminuir no caso de os blocos estarem demorando muito tempo para serem minerados. Os demais mineradores, então, começam a aplicar esforço computacional para a descoberta dos blocos subsequentes na cadeia.

O protocolo pode ser descrito conforme os passos apresentados à seguir.

1. Recuperar a **Hash** do bloco imediatamente anterior, para referenciar no novo bloco;
2. Agrupar diversas transações transmitidas na rede, formando um bloco de transações;
3. Calcular uma **Hash** para o novo bloco, utilizando os dados do cabeçalho, juntamente com um valor arbitrário (**Nonce**). A **Hash** gerada a partir dessa operação deve ser comparada com a dificuldade do sistema naquele momento;
4. Se o valor da **Hash** gerado for menor que a dificuldade, então o bloco é válido. Caso contrário, incrementa-se o **Nonce** e o passo anterior é retomado;
5. O bloco válido é propagado na rede e inserido na cadeia de blocos e o minerador é recompensado. Caso de mais de um bloco tenha sido proposto, a maior cadeia será considerada válida.

### 3.6.2 Prova de Participação

Prova de participação (PoS, do inglês *proof of stake*) surgiu como uma alternativa à prova-de-trabalho, exigindo menor esforço computacional e tornando mais rápido e barato o processo de consenso [11]. Nesse protocolo, é atribuído um validador a qualquer usuário que realizar um tipo especial de transação, onde esse usuário envia um valor  $x$  de unidade monetária que permanece retido, servindo como garantia de sua honestidade, comprovando que investiu dinheiro no sistema. Esse valor é chamado de participação e, em determinados sistemas, quanto maior a participação de um usuário, maior a chance dele ser eleito líder e possuir o direito de propor um novo bloco. Para um usuário malicioso realizar um ataque na rede, torna-se necessário que este adquira uma grande quantidade de fundos, o que pode ser impraticável (embora não impossível) devido ao alto custo financeiro e à natureza de base monetária limitada das moedas virtuais.

A cadeia de blocos mantém o registro de um número de validadores, que poderão propor novos blocos e decidirão quais serão validados, através de algoritmos de consenso tradicionais. Existem dois tipos principais de algoritmos de consenso que se relacionam com prova de participação. São eles *chain-based proof of stake* e *BFT-style proof of stake*.

- **Chain-based proof of stake:** o algoritmo seleciona aleatoriamente um validador e garante a ele o direito criar um bloco, que deve apontar a algum bloco anterior. Com o passar do tempo, a maioria dos blocos converge para uma cadeia em constante crescimento;
- **BFT-style proof of stake:** validadores são selecionados aleatoriamente e a cada um é dado o direito de propor um bloco. Então, iniciam-se múltiplas rodadas de votos, onde, a cada rodada, todo validador tem direito de votar em um bloco. No final, todos os validadores conectados decidem permanentemente quais blocos serão inseridos na Blockchain.

### 3.6.3 Practical Byzantine Fault Tolerance

Practical Byzantine Fault Tolerance [5] é um algoritmo de replicação capaz de tolerar falhas bizantinas<sup>4</sup> [14]. Protocolos de consenso tolerantes a falhas bizantinas são importantes no contexto de moedas virtuais pela capacidade de atingir consenso mesmo na presença de nodos que apresentem comportamento malicioso. Enquanto a prova-de-trabalho apresenta boa escalabilidade com desempenho fraco, protocolos como o PBFT apresentam bom desempenho para um número reduzido de nodos [26], sendo viáveis para aplicação em cadeias de blocos privadas.

Algoritmos baseados em PBFT implementam uma forma de replicação máquina de estados [13]. As réplicas avançam suas computações ao processar exatamente as mesmas requisições, recebidas em um mesma ordem. Dessa forma, cada réplica irá passar pelos mesmos estados, de forma consistente. A decisão de ordem total na entrega das requisições é garantida por um quórum de réplicas corretas, que não podem comprometer o sistema. Este quórum é determinado por  $n = 3f + 1$ , sendo  $n$  o número de réplicas e  $f$  o número máximo de réplicas maliciosas.

### 3.6.4 Prova de tempo decorrido

Prova de tempo decorrido (POeT, do inglês *proof of elapsed time*) é um algoritmo de consenso utilizado com a premissa de reduzir o custo com energia ou equipamentos especializados, utilizando instruções contidas nos processadores mais atuais [18]. No PoET, cada validador solicita um tempo de espera a uma função confiável, chamada de enclave, executada diretamente pela cadeia de blocos. Aquele que receber o menor tempo é eleito o líder, ao final da espera. Uma vez escolhido o líder, o algoritmo associa um temporizador ao bloco de transações, que serve como certificado de honestidade, enquanto os demais nodos realizam o processo de verificação das transações. Se o bloco for validado com sucesso, a enclave gera um certificado de espera, confirmado que todos os temporizadores atribuídos foram respeitados e, então, o bloco pode ser inserido na cadeia de blocos.

Esse processo pode ser ilustrado pelos seguintes passos:

1. Geração do temporizador a todos os nodos participantes;
2. Quando o primeiro temporizador se esgotar, o nodo associado a ele torna-se líder;
3. Um novo bloco é proposto, sendo as transações transmitidas neste bloco;
4. Um temporizador é associado ao bloco;
5. Se o bloco estiver correto e o temporizador do bloco estiver esgotado, continuar. Caso contrário, o processo retoma o passo 1;
6. Enclave gera um certificado de espera, confirmado a validade dos passos anteriores;
7. Bloco é inserido na cadeia de blocos.

**Observação 3.6.1 — Curiosidade.** Os conceitos utilizados em Blockchains, em sua grande maioria, são ideias de pesquisa esquecidas na literatura. Por exemplo, a ideia de *Contratos Inteligentes* foi proposta por Nick Szabo [24], em 1994, com o objetivo de permitir a implementação de código que possa ser verificável; Protocolos como a *prova de trabalho* foram propostos inicialmente como uma tentativa de evitar *spams* em servidores de e-mail. Ao enviar um email, o usuário deveria comprovar a integridade daquele email garantindo que foi efetuado um significativo esforço computacional para a produção do email, na tentativa de eliminar *spams* gerados automaticamente [8]. Para saber mais, o leitor é convidado à ler o artigo **Bitcoin's Academic Pedigree** [16].

<sup>4</sup>Um nodo pode apresentar comportamento arbitrário ou malicioso, gerando dados errados ou omitindo envios e respostas durante a troca de mensagens com outros participantes. Esse comportamento, chamado bizantino, é difícil de detectar.

### 3.7 Plataformas

Após o Bitcoin, muitas outras plataformas surgiram. Cada uma possuindo peculiaridades em seus protocolos e implementações. A seguir, citamos as mais conhecidas, mas o leitor pode encontrar outras no link: <https://coinmarketcap.com> que apresenta uma lista das criptomoedas mais cotadas no momento.

#### 3.7.1 Ethereum

O Ethereum é uma plataforma Blockchain que possui como moeda principal o Ether, que pode ser adquirido em corretoras de criptomoedas e/ou diretamente com pessoas de confiança que já possuem Ether (conhecidos como P2P). Após o Bitcoin, o Ethereum é a Blockchain mais utilizada e popularizada na comunidade, porém, esta ainda apresenta muitos problemas de escalabilidade.

Além do Ether, no Ethereum pode-se desenvolver scripts que são executados dentro da Blockchain, como um aplicativo descentralizados (DApps). Como já citado anteriormente, esses scripts são conhecidos como Contratos Inteligentes. Para fazer a criação de um Contrato Inteligente, o programador precisa ter acesso a uma interface de Deploy de contratos, e para fazer esse Deploy precisamos de uma quantidade em Ether que é utilizada como recompensa ao minerador responsável por criar esse contrato. Nos capítulos seguintes exploramos mais sobre o Ethereum.

#### 3.7.2 EOS

A EOS é uma plataforma recente (lançada em 2018) e muito semelhante ao Ethereum. Além de também possuir suporte ao desenvolvimento de Contratos Inteligentes, muitos dos conceitos do Ethereum foram replicados nela.

Para conhecer mais sobre ela, sugerimos a leitura da sua documentação, em: <https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>.

#### 3.7.3 IBM Hyperledger

Hyperledger é um projeto que visa a criação de plataformas e arcabouços para cadeias de blocos privadas, que facilitem a integração entre diversas indústrias. Criado e mantido pela The Linux Foundation<sup>5</sup>, conta com colaboradores do mercado financeiro, de cadeias de produção, Internet das Coisas, manufatura, entre outras áreas<sup>6</sup>. Segundo a página oficial do projeto Hyperledger<sup>7</sup>, a ideia é que somente um projeto de código aberto pode cumprir os requisitos necessários para que haja uma adoção em massa de sistemas comerciais baseados em cadeia de blocos.

#### 3.7.4 Hyperledger Fabric

Hyperledger Fabric é um sistema modular para cadeias de blocos privados e permissionados [27]. Em uma cadeia de blocos privada, as opções de escrita são controladas por um ente central, enquanto as opções de leitura podem ser limitadas a poucos nodos ou abertas a qualquer usuário do sistema distribuído. O fato de ser permissionado implica em maior controle sobre a identidade dos usuários da rede. De acordo com a documentação oficial<sup>8</sup>, Fabric possui um subsistema de cadeia de blocos contendo dois componentes principais: estado global e registro de transações. O estado global descreve o estado da cadeia de blocos em qualquer momento de tempo, enquanto o registro de transações grava todas as transações que resultaram no valor atual do estado global, formando um histórico de atualizações.

Diferentemente do Bitcoin, os contratos inteligentes nesse sistema são invocados por um software chamado Chaincode. Esses contratos inteligentes normalmente só interagem com o estado global e são implementados na linguagem Go. Para realizar consenso, Fabric apresenta uma implementação do *Practical Byzantine Fault Tolerance* [5].

---

<sup>5</sup>The Linux Foundation: <https://www.linuxfoundation.org>

<sup>6</sup>Hyperledger projects: <https://www.hyperledger.org/projects>

<sup>7</sup>Hyperledger: <http://hyperledger-fabric.readthedocs.io/en/latest>

<sup>8</sup>Hyperledger Fabric Documentation: <http://hyperledger-fabric.readthedocs.io/en/latest>

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### 4. Ethereum

Em 2013, Vitalik Buterin propôs a ideia de uma rede Blockchain que pudesse abrigar códigos programáveis e executar funções que expandiriam a tecnologia para outras áreas além do Bitcoin. Após captar investimentos de 31.591 Bitcoins (cotados na época em aproximadamente U\$ 18 milhões de dólares) em uma ICO que durou 42 dias, o projeto foi lançado em 2015 pelos seus fundadores Gavin Wood, Jeffrey Wilcke e o próprio Vitalik Buterin.

Usuários do Ethereum podem utilizar da sua plataforma de desenvolvimento para a criação de qualquer operação e com a complexidade desejada, não se limitando apenas a criptomoedas.

#### 4.1 Especificações

Não iremos aprofundar em todos os aspectos do Ethereum, porém alguns conceitos são fundamentais para o entendimento da plataforma e necessários para a iniciação no desenvolvimento de contratos inteligentes, entre eles:



Figura 4.1: Logo da Fundação Ethereum

##### 4.1.1 Ethereum Virtual Machine (EVM)

Em termos computacionais a EVM é uma Máquina de Estados Virtual e Turing-Completa, definida pela tupla: **(estado do bloco, código, memória, pilha, contador de programa e gas)**. Para simplificar, podemos dizer que é o ambiente utilizado para a execução de contratos inteligentes.

Isso significa que todo tipo de transação que acontece na blockchain do Ethereum ocorre dentro da EVM.

Funciona da seguinte forma: um desenvolvedor cria um Contrato por meio de uma linguagem alto-nível orientada a contratos, como Solidity, Serenity ou Viper. Esse contrato é compilado e transformado em um código de bytes (ou, EVM bytecode) que então é executado na Ethereum Virtual Machine.

A EVM gerencia diferentes tipos de dados dependendo do seu contexto. Entre os principais tipos de dados, podemos citar: *stack*, *calldata*, *memory* e *storage*. Cada um desses tipos exige um tratamento específico pela EVM, e nos capítulos subsequentes veremos como fazer operações com esses dados através da linguagem Solidity.

### 4.1.2 Contas

No Ethereum, existem dois tipos de contas: as **Contas de Domínio Externo** (do inglês, Externally Owned Accounts, ou EOAs) e as **Contas de Contrato**. Podemos abstrair uma *EOA* como uma conta bancária de pessoa física, nela temos um saldo em Ether, podemos realizar transações com esse Ether e a controlamos por uma chave privada.

Já as contas de contrato são um pouco mais complexas do que uma conta jurídica, elas também possuem um saldo em Ether, entretanto, além disso, elas possuem um **código** associado. Esse código é executado (ou ativado) toda vez que um gatilho desse contrato é disparado por outra conta, através de uma transação ou uma mensagem, e, é a partir desse código - denominado Contrato Inteligente - que podemos explorar as propriedades de armazenamento e processamento do Ethereum.

Devemos imaginar uma conta de contrato como um agente autônomo que vive na rede. Assim que ela é criada, o seu código associado estará executando de forma **ininterrupta** seguindo a lógica em que foi escrito.

**Observação 4.1.1 — Analogia.** Se removermos as **Contas de Contrato** do Ethereum, o resultado seria uma rede semelhante ao Bitcoin, onde haveria apenas usuários fazendo a “troca” de um ativo digital (nesse caso, o Ether). Portanto, a inovação do Ethereum é originado da possibilidade de criação de contas associadas a **Contratos Inteligentes**.

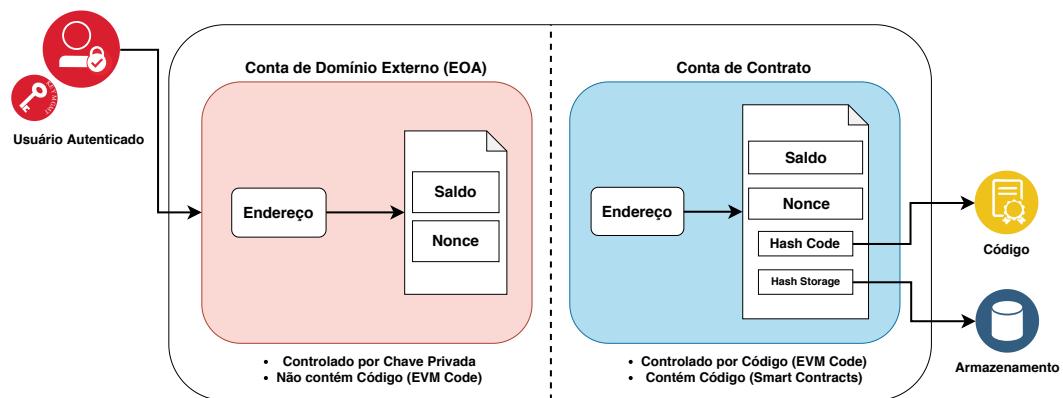


Figura 4.2: Tipos de Contas no Ethereum [19]

### 4.1.3 Transações e Mensagens

Segundo a documentação do Ethereum, uma transação é um pacote de dados que será enviado de uma conta EOA para outra conta EOA, ou para uma conta de contrato. Em resumo, **uma transação é a maneira como os indivíduos (EOA) se comunicam pela rede Ethereum**. Sua estrutura é a seguinte:

- **Nonce:** número de transações enviadas pelo remetente.

- **To:** campo para um endereço de 20 bytes que representa o **destinatário** da transação. Pode ser uma EOA, uma conta de contrato ou vazio.
- **v, r, s:** campos que correspondem a assinatura da transação, usados para representar o **remetente** da transação.
- **Value:** campo de valor escalar que representa a quantidade que um indivíduo está enviando para outro indivíduo (transferência de fundos) ou para um contrato (depósito de fundos em um contrato).
- **Data:** campo opcional e de tamanho ilimitado que representa um dado que o indivíduo deseja passar para um contrato. É a maneira que um EOA pode fazer a chamada de uma função em um contrato.
- **GasLimit:** número máximo de unidades computacionais que o remetente está disposto a pagar para executar uma transação (em gas).
- **GasPrice:** valor que o remetente está disposto a pagar pela execução de cada unidade computacional (em wei).

Mensagens são objetos virtuais que existem no ambiente de execução do Ethereum e que são originadas de chamadas de funções. É a maneira como um contrato se comunica com outro contrato ou com uma EOA, dentro da rede Ethereum. A sua diferença em relação a uma transação é que as mensagens são produzidas pelos próprios contratos e não pelos usuários EOA. Sua estrutura é bem semelhante a das transações.<sup>1</sup>

#### 4.1.4 Gas

Sempre que um contrato é executado após ser chamado por uma transação ou mensagem, todas as instruções de máquina são executadas em todos os nós da rede Blockchain (lembre-se que falamos de uma rede ponto-a-ponto). Essa execução exige um custo computacional (energético, banda, desgaste da máquina, etc.) dos nós, portanto, foi definido o **gas** como a unidade de medição desse custo.

Todo fragmento de computação programada no Ethereum, cobrará um imposto (**fee**) em gas daquele que requisitar essa computação. Isso inclui: criação de contratos, chamadas de mensagem, uso e acesso do armazenamento de contas e execução de operação na EVM. Esse imposto tem duas funções principais: recompensar os membros da rede (nós) pela computação cedida e também forçar que os usuários sejam mais responsáveis no uso da rede (consequentemente evitando ataques maliciosos).

Como mostramos anteriormente, toda transação exige os campos: GasLimit que é a unidade máxima em gas que um usuário está disposto a pagar pela transação, e um valor GasPrice que é o valor que o usuário julga pertinente para a unidade de gas. O imposto final é calculado pela fórmula:

$$\text{Teorema 4.1.2} \quad \text{Fee} = \text{GasUsed} \times \text{GasPrice}$$

Onde o GasUsed é um valor inteiro que representa a quantidade de gas realmente consumida na transação e que deve ser menor que o GasLimit. O GasPrice é medido em **wei** que é a menor unidade do Ether (como o centavo é para o real). Um valor alto de GasPrice pode fazer com que sua transação seja validada mais rapidamente, pois os mineradores aumentam a “preferência” por transações que são mais valiosas para eles.

Esse valor é obtido do cálculo de todas as instruções de máquina utilizadas para realizar aquela operação, através do que chamamos de **opcode** (ou, código de operação). Na tabela 4.1 apresentamos alguns exemplos dessas instruções e seus respectivos valores em gas, contudo, esses valores frequentemente são ajustados de acordo com o propósito das atualizações do Ethereum.

**Observação 4.1.3 — Alerta.** Caso o GasLimit indicado por um usuário seja menor que o GasUsed consumido na transação, o sistema irá disparar o erro **Out of Gas**, onde a transação não será executada e ele perderá todo o seu valor calculado por  $\text{Fee} = \text{GasLimit} \times \text{GasPrice}$ . Outro caso possível, é quando um usuário insere um valor GasPrice muito baixo e nenhum minerador aceita fazer a validação dessa transação, nesse caso nenhum fundo é perdido mas a

<sup>1</sup>Vide Referências: [29]

Opcode	Valor (Gas)	Descrição
ADD	3	Adição de Valores
SUB	3	Subtração de Valores
MUL	5	Multiplicação de Valores
CREATE	32000	Criação de Conta com Código Associado
STORE	[5000..20000]	Armazenamento de Dados (até 32 bytes)

Tabela 4.1: Exemplos de Instruções e Seus Valores em Gas

transação não será executada.

O **ETH Gas Station** é uma ferramenta interessante que pode nos auxiliar no cálculo dos parâmetros adequados de transações. Pode ser encontrado em [ethgasstation.info](http://ethgasstation.info).

#### 4.1.5 Ether

O Ether – representado por ETH – é a unidade base do Ethereum. Além de ser utilizado como moeda digital, ele possui a função de **recompensar os membros da rede que são responsáveis por validar os blocos que contém transações e mensagens**.

Sempre que um conjunto (ou bloco) de transações e mensagens é inserido na Blockchain, o nó responsável por fazer a validação desse conjunto recebe duas recompensas:

- a. Uma taxa fixa de 2 ETH que é emitida pela própria rede (daqui vem o termo mineração).
- b. Uma taxa variável de Fee que foi cobrada dos emissores das transações.

Dessa forma, as maneiras possíveis de se obter Ethers são: participando da rede como um nó validador, ou comprando de mineradores que já receberam essas recompensas.

**Observação 4.1.4 — Curiosidades.** Na verdade, o cálculo da taxa variável de mineração é feito por uma fórmula matemática um pouco mais complexa, que depende de parâmetros internos da rede. Já a taxa fixa, antes da atualização Constantinopla, era de 3 ETH, mas foi alterada para 2 ETH como forma de valorização da moeda. Vide documentação para mais detalhes.

Apesar de possuir um valor de mercado relativamente alto, o Ether pode ser fracionado em unidades menores. O wei, unidade utilizada no cálculo de *Fee* é a unidade atômica do Ether, e representa  $10^{-18}$  ether. As principais unidades utilizadas estão representadas na tabela 4.2.

Unidade	Valor (ETH)
Ether	1
Finney	$10^{-3}$
Szabo	$10^{-6}$
Wei	$10^{-18}$

Tabela 4.2: Principais Unidades Fracionadas do Ether

Para os interessados em obter ETH para qualquer que seja o propósito, deve-se inicialmente obter uma carteira, que será a responsável por criar uma conta **EOA**. A carteira irá fornecer sua chave privada (que é o único meio de acesso ao controle dessa conta) e o seu **endereço**. Então, através desse endereço, estará apto a receber ETH tanto por mineração, quanto pela aquisição via corretoras ou indivíduos.

Atualmente, a carteira mais popular é a **Metamask**, uma carteira alocada em Browser (Chrome, Firefox ou Brave) via extensão, que fornece interfaces de comunicação com aplicações descentralizadas no Ethereum (chamadas DApps). Isso significa que o usuário poderá executar aplicações do Ethereum – como jogos, sistemas de apostas e votos, etc. – sem o uso de qualquer software em Desktop.

Porém, em termos de armazenamento, essa provavelmente não é a escolha mais segura, visto que aplicações em Browser podem oferecer inúmeras vulnerabilidades. Nesse caso, uma melhor opção seria a utilização de **Cold Wallets**, que são carteiras armazenadas em sistemas sem acesso a Internet, semelhante a um pendrive.

#### 4.1.6 Bloco

No Ethereum, o **Bloco** é a união de uma coleção de dados importantes (*block Header*) com um conjunto de transações e mensagens, e informações sobre os blocos anteriores a ele. Alguns dos dados importantes de um bloco que podemos citar, são:

1. **Number:** índice daquele bloco em relação a toda a cadeia de blocos.
2. **ParentHash:** hash 256-bit de todo o bloco anterior a esse.
3. **Nonce:** valor 64-bit único que comprova a quantidade de computação que foi necessária para minerar esse bloco.
4. **Beneficiary:** endereço da pessoa responsável por minerar esse bloco e que receberá a recompensa em ETH.
5. **Reward:** valor total em ETH da recompensa que o Beneficiary receberá.
6. **Difficulty:** valor escalar que regula a dificuldade de se minerar aquele bloco. Geralmente é controlado pela equipe Ethereum para regular a velocidade de mineração dos blocos.
7. **Timestamp:** valor escalar que representa uma data e hora calculada pela fundação Unix.
8. **GasLimit:** valor máximo que pode ser consumido pelas transações e mensagens dentro daquele bloco (geralmente é 8.000.000).
9. **GasUsed:** quantidade total de gas usado pelas transações e mensagens naquele bloco.
10. **Size:** quantidade em bytes utilizada pelas transações e mensagens naquele bloco (calculada pelo GasUsed).

⑦ Block Height:	8280294	< >
⑦ Timestamp:	① 1 min ago (Aug-03-2019 10:03:17 PM +UTC)	
⑦ Transactions:	23 transactions	and 5 contract internal transactions in this block
⑦ Mined by:	0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5 (Nanopool)	in 7 secs
⑦ Block Reward:	2.008235443041677692 Ether (2 + 0.008235443041677692)	
⑦ Uncles Reward:	0	
⑦ Difficulty:	2,234,145,703,152,098	
⑦ Total Difficulty:	11,302,407,168,871,278,604,725	
⑦ Size:	17,862 bytes	
⑦ Gas Used:	7,928,490 (99.06%)	
⑦ Gas Limit:	8,003,840	
⑦ Extra Data:	PPYE nanopool.org (Hex:0x50505945206e616e6f706f6f6c2e6f7267)	

Figura 4.3: Informações de um Bloco do Ethereum

Uma maneira interessante de acompanhar a inclusão de novos blocos na cadeia sem a instalação da rede Ethereum, é através do site [etherscan.io](https://etherscan.io), nele, além de obter todas as informações referente aos blocos em tempo real, também é possível visualizar as transações e mensagens que estão relacionadas aquele bloco e obter dados estatísticos da rede.

#### 4.1.7 Consenso: PoW vs PoS

O Ethereum segue um roteiro (ou *roadmap*) que define o desenvolvimento da rede como um todo. Atualmente, o Ethereum usa como protocolo de consenso o algoritmo denominado Proof-of-Work (o mesmo utilizado pelo Bitcoin), contudo, nesse roteiro é previsto uma lenta migração para outra técnica, denominada Proof-of-Stake (já visto anteriormente).

Apesar de não ser a única, essa é a principal evolução proposta para a futura plataforma - batizada de *Ethereum 2.0* - e, a conclusão dessa migração está prevista para o final de 2019 com a atualização **Serenity**.

**Observação 4.1.5 — Recomendação.** Para maior aprofundamento no algoritmo de consenso Proof-of-stake, recomendamos a leitura do artigo **PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake**, o primeiro a definir a técnica.

#### 4.1.8 Documentação e Ferramentas

O Ethereum possui três documentações fundamentais, que foram utilizadas como referências principais nesse capítulo, e que indicamos ao leitor:

1. **White Paper:** Documentação com visão mais alto-nível da plataforma, com explicações sucintas, de linguagem facilitada e com exemplos voltados ao mundo real. Para aqueles com pouco conhecimento computacional, é a mais recomendada. Encontrada em: [github.com/ethereum/wiki/wiki/White-Paper](https://github.com/ethereum/wiki/wiki/White-Paper) [3]
2. **Beige Paper:** É uma versão mais “light” do yellow paper, nela estão explicadas alguns dos conceitos fundamentais do Ethereum de maneira mais conceitual e algorítmica. Encontrada em: [github.com/chronaeon/beigepaper/blob/master/beigepaper.pdf](https://github.com/chronaeon/beigepaper/blob/master/beigepaper.pdf)
3. **Yellow Paper:** Documentação mais baixo-nível, nela estão especificadas fórmulas matemáticas que explicam as complexidades do funcionamento do Ethereum. Encontrada em: [ethereum.github.io/yellowpaper/paper.pdf](https://ethereum.github.io/yellowpaper/paper.pdf)

Para quem deseja trabalhar com o desenvolvimento *full-stack* de aplicações distribuídas no Ethereum, atualmente existem diversas ferramentas criadas para auxiliar nesse trajeto. Na tabela 4.3 citamos as mais populares, pode-se encontrar a lista mais completa no site: [ethereum.org/developers](https://ethereum.org/developers).

Propósito	Linguagem de Contrato	Ferramenta de Desenvolvimento	IDE	API (Frontend)	Explorador de Blocos
Ferramenta	Solidity, Vyper, LLL.	Truffle, Embark, Waffle.	Remix, VS Code, Superblocks.	web3.js, ethers.js, light.js.	Etherscan, Etherchain.

Tabela 4.3: Principais Ferramentas Para Desenvolvimento no Ethereum

## 4.2 DApps e Contratos Inteligentes

Aplicativos Descentralizados (anteriormente conhecidos como **Organizações Autônomas Descentralizadas**) são aplicações que executam em redes ponto-a-ponto. Apesar de existirem há muitos anos (BitTorrent, Tor, BitMessage, etc.) o termo foi difundido apenas após a ascensão das redes Blockchain, em especial, o Ethereum.

No artigo "**The General Theory of Decentralized Applications, DApps**" os autores propõem algumas características principais dos DApps são:

- São de código aberto e operam de maneira autônoma e ininterrupta.
- Geram **tokens** a partir de um algoritmo padrão, que são responsáveis por fomentar o uso da aplicação.
- A aplicação pode adaptar seus protocolos em resposta aos feedbacks dos usuários, desde que seja decidido por consenso.

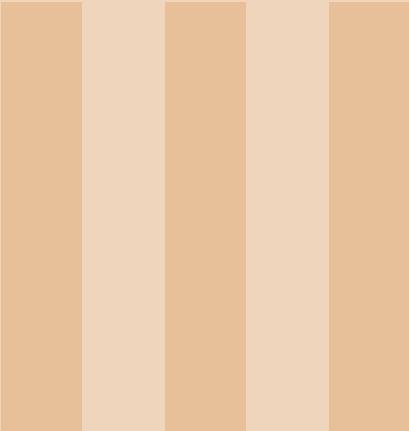
Um DApp envolve toda a “pilha” de desenvolvimento de uma aplicação: a infraestrutura, o back-end, o front-end, as interfaces de comunicação, etc., da mesma forma que uma aplicação centralizada. A novidade, está na existência dos **Contratos Inteligentes** (do inglês, Smart Contracts).

São esses contratos que estão de, de fato, executando na Blockchain, fazendo uso de seus recursos e explorando suas particularidades, e, por esse motivo, é de grande importância que sejam elaborados de forma correta e robusta.

```
1 pragma solidity >=0.4.0 <0.7.0;
2
3 contract SimpleStorage {
4     uint storedData;
5
6     function set(uint x) public {
7         storedData = x;
8     }
9
10    function get() public view returns (uint) {
11        return storedData;
12    }
13 }
```

Figura 4.4: Exemplo de Contrato Inteligente Escrito em Solidity

Na figura 4.4 temos o exemplo de um contrato inteligente que recebe um valor “x” e o armazena na variável “storedData”, que posteriormente será gravada na Blockchain de forma persistente. Nos próximos capítulos aprofundaremos na escrita desses contratos através do uso da Linguagem Solidity, utilizando algumas ferramentas de desenvolvimento do Ethereum.



# Colocando em Prática

<b>5</b>	<b>Desenvolvendo Contratos Inteligentes</b>	<b>36</b>
5.1	Mais Contratos Inteligentes	
5.2	Ambiente de Desenvolvimento: Remix	
5.3	Solidity	
<b>6</b>	<b>Estudo Guiado</b>	<b>46</b>
6.1	<i>Ethereum Improvement Proposals (EIP)</i>	
6.2	Criptomoeda e Token	
6.3	Criando nosso Token	
<b>7</b>	<b>Ferramentas Utilizadas</b>	<b>51</b>
7.1	Ferramentas e Notícias	
7.2	Notícias	
	<b>Bibliography</b>	<b>53</b>
	Artigos	
	Livros	
	Outros	
	<b>Index</b>	<b>55</b>

# APOSTILA PARA INICIAINTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### 5. Desenvolvendo Contratos Inteligentes

#### 5.1 Mais Contratos Inteligentes

Como já citamos anteriormente, existem inúmeras plataformas para criação de Contratos Inteligentes, entre elas: Ethereum, Hyperledger Fabric, Stellar, EOA, e outras. Os motivos de escolhermos o **Ethereum** como estudo são simples: é a plataforma mais popular, com documentação mais ampla e, atualmente é a plataforma com maior relevância no mercado e no meio acadêmico.

Nesse capítulo, inicialmente, exploramos os conceitos básicos da ferramenta **Remix**, que será utilizada no desenvolvimento dos Contratos Inteligentes. Em seguida, apresentamos o básico da linguagem **Solidity** (versão 0.5.1), uma linguagem de programação de contratos feita especialmente para uso na rede Ethereum, mas que já é explorada por outras redes.

#### 5.2 Ambiente de Desenvolvimento: Remix

O Remix é uma “super-IDE” criado pelo grupo Ethereum para facilitar o desenvolvimento de dApps e Contratos Inteligentes para a plataforma. A principal característica (e motivo de o usarmos nessa apostila) é ser um ambiente alojado em Navegador, ou seja, não precisamos baixar ou instalar nenhum programa ou biblioteca adicional.

Já os motivos que nos levam a chamá-lo de “super” são bem claros: além de possuir as ferramentas básicas para desenvolvimentos, como editor de texto, compilador e debugger, o Remix também possui algumas ferramentas interessantes. Sendo elas, um gerador de carteiras e quantias de Ethers em um ambiente de testes virtual (denominado Javascript VM) que nos permitem fazer uma análise total de custos dos contratos e, também, acessar todos os Opcodes usados na criação e execução dos seus contratos.

A figura 4.1 mostra a interface básica do Remix e acessando esse link: [remix.ethereum.org](https://remix.ethereum.org) você será redirecionado para o Remix.

1. No menu lateral à esquerda ficam as ferramentas principais: compilador, deployer, analisador sintático, gerador de testes, etc.
2. Aqui fica o explorador de arquivos. Os arquivos de código Solidity, são salvos com o sufixo .sol.
3. Nesse quadrante central, é onde trabalharemos com o editor de texto. Clicando no botão "Solidity", o Remix irá automaticamente configurar as ferramentas necessárias para uso da linguagem Solidity. Esse processo pode ser feito manualmente acessando a aba “Plugin Manager” e escolhendo as ferramentas desejadas.

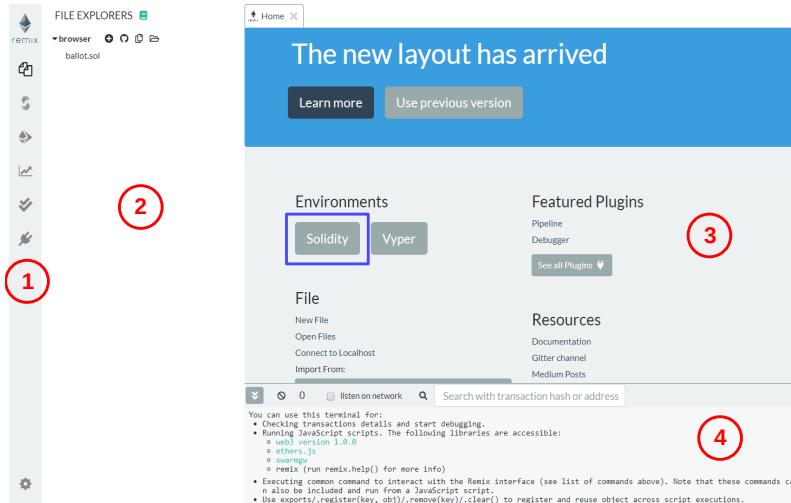


Figura 5.1: Interface do Remix

4. Na barra inferior temos um console com outputs e informações gerais de compilação.

**Observação 5.2.1 — Nota.** Nessa apostila usamos a interface mais recente do Remix (lançada em 2019), porém, nas configurações é possível escolher por uma versão mais limpa e com poucos efeitos visuais.

### 5.2.1 Criação de Arquivo

No Solidity, a principal entidade que trabalharemos é o **contract**. Podemos abstraí-lo como sendo o equivalente da **class** em C++. Utilizaremos o exemplo abaixo para explicar o uso do Remix, não se assuste, pois, nas seções posteriores entenderemos mais sobre a sintaxe da linguagem.

```

1 // cabecalho do contrato: aqui definimos a versao do compilador
2 pragma solidity ^0.5.1;
3
4 // definindo um contrato
5 contract MeuPrimeiroContrato{
6     string public nome; // declarando um atributo 'nome' do tipo string
7     uint private idade; // declarando um atributo 'idade' do tipo uint
8
9     // construtor do contrato
10    constructor () public{
11        nome = "Maria";
12        idade = 20;
13    }
14
15    // declarando uma funcao de atribuicao
16    function setNome(string memory _nomeIN) public{
17        nome = _nomeIN;
18    }
19
20    // declarando uma funcao de retorno
21    function getName() public view returns (string memory){
22        return nome;
23    }
24 }
```

O primeiro passo é acessar o **Explorador de Arquivos**, criar um novo arquivo com o pósfixo .sol e adicionar o código. Do contrato anterior, podemos antecipar alguns elementos:

- Comentários no código são feitos utilizando “//” ou entre “/\* \*/”, semelhante a C/C++ e outras linguagens populares.
- O construtor de um contrato é declarado com a palavra reservada **constructor**, não possui nome e pode ser do tipo **public** ou **internal**.

- Funções são declaradas com a palavra reservada **function**, semelhante a declaração em Javascript, seguidas por um *nome (parâmetros) modificadores* e em caso de retorno utilizar-se da palavra reservada **returns** (*tipo nome* da variável retornada).
- Os código em Solidity não possuem uma função “main” para instanciar contratos, objetos ou chamar funções. Esses eventos são realizados pelas **transações** ou **mensagens**. Exemplificamos mais a frente.

### 5.2.2 Compilação e Deploy

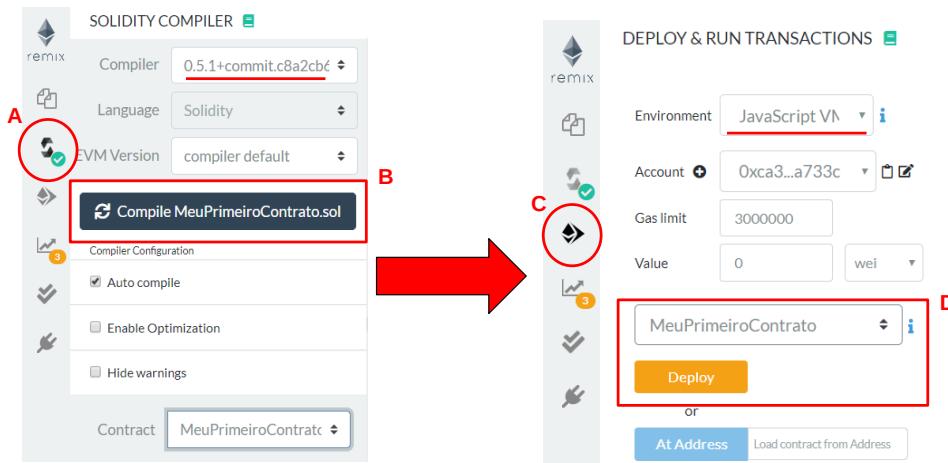


Figura 5.2: Compilando e Criando Nosso Primeiro Contrato

Ao longo dessa apostila, sempre que criarmos um novo contrato ou implementarmos novas funcionalidades, seguiremos essa etapa básica de compilação e criação do contrato, sem uso de terminal. Portanto, sempre que usarmos o termo **Compilação e Deploy**, esses passos devem ser seguidos (veja Figura 5.2).

- Acessamos a aba “Solidity Compiler” pelo menu lateral.
- No botão **Compile NomeContrato** fazemos a compilação do respectivo contrato. Selecionando a opção **auto compile** essa etapa será suprimida futuramente.
- Acessamos a aba “Deploy & Run Transactions” pelo menu lateral.
- Selecionamos o respectivo contrato e no botão **Deploy** fazemos a sua criação. Tenha certeza de selecionar a “JavaScript VM” como ambiente de teste.

### 5.2.3 Acessando Métodos e Atributos

Após o Deploy do contrato, seremos capazes de acessar seus métodos e atributos. A interface é intuitiva, e o método pode ser descrito da seguinte forma (veja Figura 5.3):

- Expandimos os métodos e atributos do contrato já instanciado na seta lateral. Aqui também podemos acessar o **endereço de contrato** do nosso contrato (nesse exemplo, 0x692...77b) que é gerado após o deploy do mesmo.
- Após expandir o contrato, podemos acessar os métodos de acordo com o seu escopo (nesse caso, apenas os que estão modificados como **public** ou **external**). A interface irá retornar as saídas.
- Para executarmos a função *setNome*, precisamos chamar uma transação. Para isso, basta inserir o valor de entrada no local indicado.
- Em seguida, executamos a transação. Chamando novamente a função *get*, veremos que o valor foi modificado.

### 5.2.4 Analisando Transações e Chamadas de Funções

Após executar alguma transação, ou acessar métodos/atributos de um contrato, é possível fazer a análise da execução ocorrida. Para isso, acessamos o **Console** (interface 4, veja Figura 5.1) do Remix

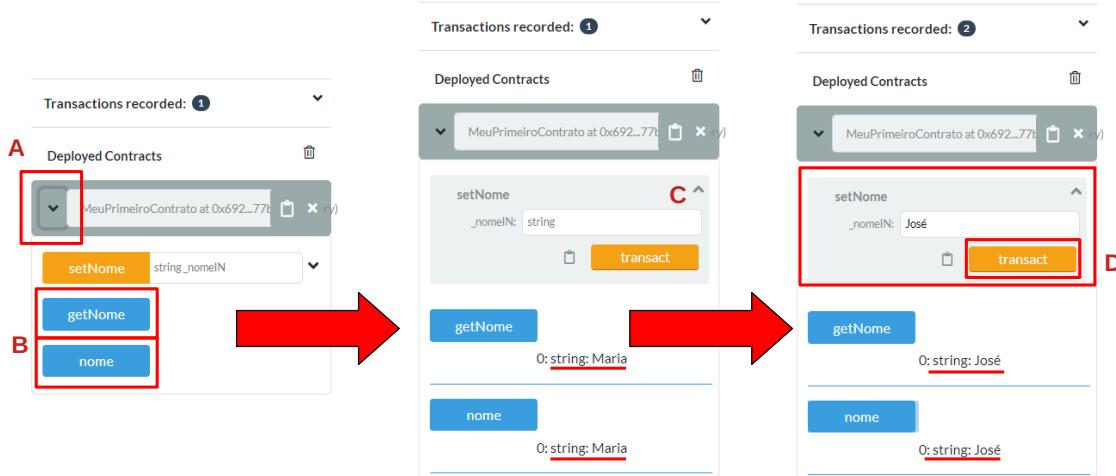


Figura 5.3: Acessando Métodos e Atributos do Contrato

e selecionamos a execução desejada (seta lateral direita, veja Figura 5.4). Algumas informações que podemos visualizar:

- **status** da transação
- **contract address**, que é o endereço do contrato compilado
- **from** é o endereço da conta que fez a chamada da execução (nesse caso é uma EOA de endereço 0xca35...33c).
- **to** é o contrato que recebeu a chamada
- **execution cost** e **transaction cost** são os custos em gas cobrados por essa transação. Para entender melhor a diferença entre eles, recomendamos o artigo: <https://vomtom.at/what-exactly-is-the-gas-limit-and-the-gas-price-in-ethereum>.
- Outros dados são menos relevantes para nós no momento, porém, são de grande importância para o armazenamento das transações nos **blocos** da Blockchain.

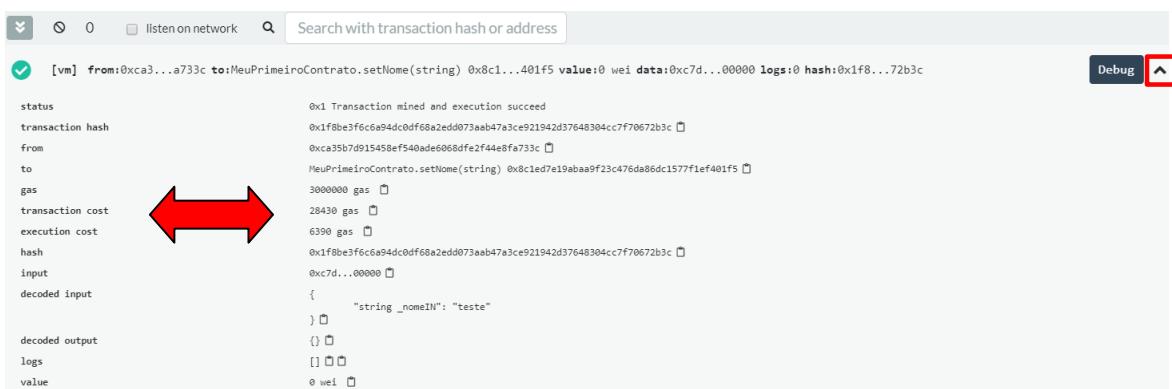


Figura 5.4: Análise de Transação Executada

### 5.2.5 Contas EOA para Chamada de Transações

Como padrão, o Remix nos fornece 5 contas **EOA** na rede JavaScript VM, que é uma Blockchain de testes. Porém, caso necessário, podemos adicionar inúmeras outras contas pela aba “Deploy and Run Transactions”, clicando no menu **(+ Accounts** (Figura 5.5). Se você se perguntava de onde surgiam os endereços **from** das transações, ali estão.

O saldo inicial de todas essas conta é por padrão 100 ETH. Sempre que executar uma transação, através da chamada de uma transação, você verá o saldo em Ether da conta selecionada reduzir

## DEPLOY &amp; RUN TRANSACTIONS

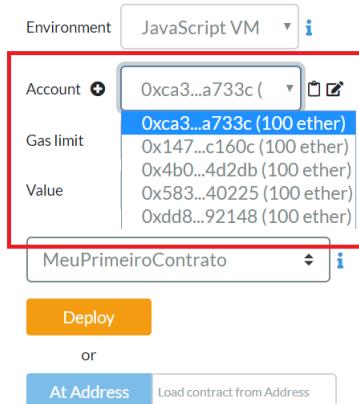


Figura 5.5: Contas EOA Fornecidas Pelo Remix

de acordo com a quantidade de gas consumido.

### 5.2.6 Gerando Saídas em JSON

JSON é um formato de representação dados utilizado por sistemas. Por ser muito leve e de fácil entendimento, se tornou muito popular em aplicações e serviços Web. No Remix, podemos acessar conjuntos de transações executadas (output) no formato JSON, de maneira simples. Para simplificar o entendimento da apostila e dos contratos implementados a seguir, todas as **saídas e casos de teste** estarão nesse formato.

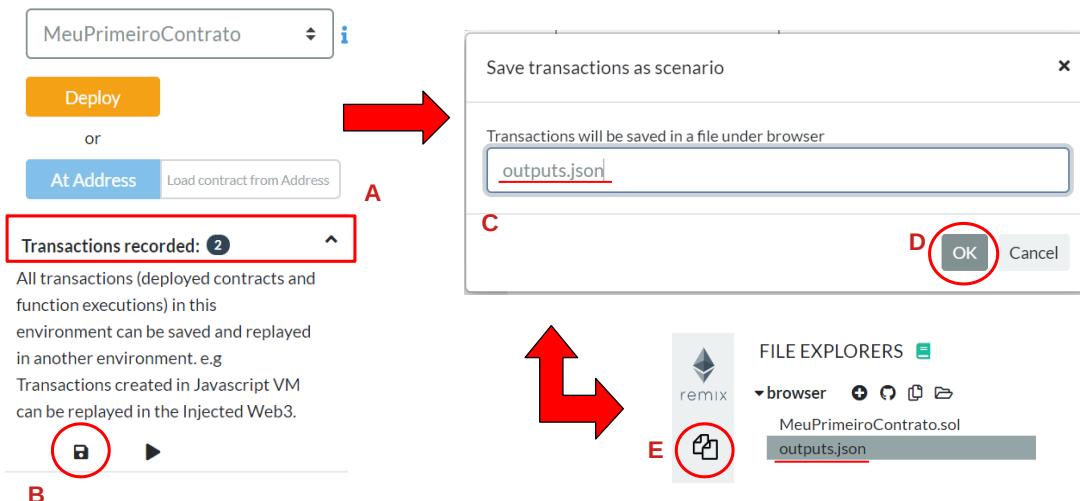


Figura 5.6: Obtendo Saídas JSON Pelo Remix

O passo a passo é simples (veja Figura 5.6):

- Expandimos a aba “Transactions Recorded”.
- Acessamos o ícone de “Save Transactions”.
- Escolhemos um nome para o arquivo com extensão **.json**.
- Salvamos o arquivo.
- Acessamos o explorador de arquivos, e lá estará nossas saídas.

Vale ressaltar que a execução de chamadas de funções que fazem apenas a leitura (ou retorno) de um dado delimitado com a palavra **memory**, não será registrada como uma transação pelo

Remix, mas como uma **Call**, que não ficará registrada nas saídas do arquivo JSON. Para visualizar essas chamadas, basta expandi-las pelo console.

**Observação 5.2.2 — Opinião.** Apesar de usarmos esse processo para melhorar o entendimento da apostila, recomendamos que o leitor **iniciante** faça o processo mais simples (Figura 5.4) através do console do Remix. As saídas serão as mesmas, porém, o acesso é mais rápido e o formato visual é mais agradável.

Após dominar os fundamentos da ferramenta, iniciaremos na linguagem propriamente dita. A documentação completa do Remix pode ser encontrada em: <https://remix.readthedocs.io/en/stable/>.

## 5.3 Solidity

Solidity é uma linguagem de alto-nível, orientada a contratos e criada para o desenvolvimento de *Smart Contracts*. Sua sintaxe possui elementos bem semelhantes a **C++** e **Javascript** (segue o padrão ECMAScript). Algumas características da linguagem:

- Tipagem estática (veja tabela 5.1)
- Superta herança múltipla e tipos complexos
- Códigos são compilados para o formato EVM-Bytecode

Tipos	Descrição
bool	“true” ou “false”
int, int8...int256	Números inteiros, pode-se declarar o tamanho em bits
uint, uint8...uint256	Números inteiros sem sinal, pode-se declarar o tamanho em bits
bytesl...bytes32	Sequência de bytes (ou vetores) declara-se o tamanho em bytes
bytes, string	Sequência de bytes de tamanho dinâmico
enum	Conjunto de tipos que serão escolhidos pelo usuário
address	Endereço Ethereum de tamanho 20 bytes

Tabela 5.1: Principais Tipos de Variáveis em Solidity

**Observação 5.3.1 — Nota.** Os exemplos nessa apostila foram implementados na versão *0.5.1* do Solidity, contudo, a velocidade de lançamento de novas versões é muito alta. Por isso, não se assuste caso a versão vigente no momento de leitura do material seja outra, pois as principais mudanças são feitas em nível de instruções, o que em grande parte, é abstruído dos desenvolvedores.

### 5.3.1 “Hello World!”

Solidity foi desenhado para se comunicar diretamente com aplicações feitas em Javascript. Pense no Solidity como uma linguagem um pouco mais complexa que **SQL**, onde estabelecemos condições aos dados e os armazenamos em um banco de dados (nesse caso, o banco é a própria Blockchain) caso as condições sejam atingidas. Por esse motivo, não há funções de output semelhantes a *printf*, *cout*, *echo* ou *alert*.

Então, uma maneira alternativa de criarmos nosso querido *Hello World!* é a partir de uma função **Getter**. Funções Getter (funções de retorno), são utilizadas apenas para leitura de dados, por isso utilizamos a palavra reservada **view** na função. Assim, obtemos:

1 `pragma solidity ^0.5.1;`  
2

```

3 contract MeuPrimeiroContrato{
4     string private ola = "Olá Mundo!";
5
6     function getOla() public view returns (string memory){
7         return ola;
8     }
9 }
```

Executando os procedimentos de deploy e execução, obtemos:

```

1 {
2     "0": "string: Olá Mundo!"
3 }
```

### 5.3.2 Estruturas de Controle de Fluxo

Em Solidity, as estruturas **if**, **else**, **while**, **do**, **for**, **break**, **continue** e **return** são declaradas da mesma forma que linguagens como C/C++ e Javascript. Entretanto, devemos tomar cuidado com algumas questões:

1. Sempre que executamos uma função, consumimos **gas**, que possui valor monetário. Portanto, loops infinitos e recursividade podem gerar custos infinitos no seu código.
2. A estrutura **if** não possui conversão de valor booleanos, portanto, a prática: **if (1) { ... }** não é válida.
3. Em geral, o uso do **if** pode ser muito custoso e possuir vulnerabilidades. Por isso, as expressões: **assert**, **require** e **revert** foram inseridas.

Um exemplo é apresentado a seguir.

```

1 pragma solidity ^0.5.1;
2
3 contract TesteThrows {
4     // as funções não acessam o storage do contrato
5     // por isso são declaradas como "pure"
6
7     function testeAssert() public pure {
8         assert(1 == 2);
9     }
10    function testeRequire() public pure {
11        require(1 == 2);
12    }
13    function testeRevert() public pure {
14        if (1 == 2){
15            revert ("Erro");
16        }
17    }
18 }
```

- **Assert**: caso a condição não seja cumprida, um erro de opcode inválido é retornado e todo o gas das operações é consumido. Use para a verificação de erros internos, como índices inválidos em vetores, divisões por zero, etc.
- **Require**: caso a condição não seja cumprida, toda operação feita é revertida e o gas consumido é devolvido. Use para verificação de erros externos, como entradas inválidas, receber Ether de outro contrato, etc.
- **Revert**: geralmente é utilizado com a estrutura **if**, quando esta não é cumprida a função revert desfaz a operação e retorna um erro para o remetente.

### 5.3.3 Herança

Segundo a documentação oficial, a linguagem possui suporte para herança e herança múltipla, incluindo polimorfismo.

Quando criamos um contrato derivado de outro contrato, apenas um contrato é criado na Blockchain, e esse possuirá o contrato base compilado dentro de si (podemos confirmar esse fenômeno quando fazemos o deploy de um contrato derivado de outro, observando o alto valor do custo de gas na instância).

Para declarar a herança, utilizamos a palavra reservada **is**, como no exemplo a seguir.

```

1 pragma solidity ^0.5.1;
2
3 contract Pessoa{
4     function anda() public pure returns (string memory){
5         return "Andando";
6     }
7 }
8
9 contract Aluno is Pessoa{
10     function assisteAula() public pure returns (string memory){
11         return "Assistindo Aula";
12     }
13 }
```

Fazendo a compilação e deploy do contrato **Aluno**, podemos fazer a chamada das duas funções **anda** e **assisteAula**, que nos retornam a seguinte saída:

```

1 {
2     "0": "string: Andando"
3 }
4 {
5     "0": "string: Assistindo Aula"
6 }
```

#### 5.3.4 Funções Modifiers

São funções utilizadas para alterar o fluxo de execução de outra função de acordo com uma condição. Não confundir com **modificadores** que servem para indicar o nível de acesso de uma função/variável.

Trouxemos um exemplo adaptado da documentação oficial do Solidity (disponível no site: <https://solidity.readthedocs.io/en/v0.5.1/structure-of-a-contract.html>):

```

1 pragma solidity ^0.5.1;
2
3 contract Purchase {
4     address public seller;
5
6     // aqui declaramos o modifier
7     modifier onlySeller() {
8         require(msg.sender == seller);
9         -
10    }
11
12     // o que esta dentro da função só sera executado
13     // caso passe da condição require no modifier
14     function abort() public view onlySeller {
15         ...
16    }
17 }
```

O operador “\_” é utilizado para indicar o ponto onde a função deve voltar a ser executada após a condição aplicada pelo modifier. Esquecê-lo pode gerar muitos problemas no seu código.

#### 5.3.5 Unidades e Variáveis Globais

##### Unidades de Ether

Um numeral pode receber os sufixos: **wei**, **finney**, **szabo** ou **ether** que representam seus respectivos valores decimais em wei.

- 1 wei == 1
- 1 szabo ==  $1.e^{12}$
- 1 finney ==  $1.e^{15}$
- 1 ether ==  $1.e^{18}$

##### Unidades de Tempo

Os sufixos: **seconds**, **minutes**, **hours**, **days** e **weeks** representam unidades de tempo em segundos. Eles não podem ser aplicados em variáveis, apenas em valores inteiros especificados.

- 1 == 1 seconds
- 1 minutes == 60 seconds
- 1 hours == 60 minutes
- 1 days == 24 hours
- 1 weeks == 7 days

Fique atento na utilização dessas unidades, pois nem todo dia possui 24 horas e nem todos anos possuem 365 dias.

### Variáveis Globais

Sempre que uma mensagem externa é enviada para nosso contrato, podemos obter informações a respeito dessa mensagem através de variáveis, sendo as principais:

- **msg.data** (bytes): a informação contida nessa mensagem.
- **msg.sender** (address): o endereço que enviou essa mensagem.
- **msg.value** (uint): o valor em wei que está sendo enviado com a mensagem.

Também podemos obter informações a respeito do **bloco** atual da rede Ethereum através das variáveis:

- **blockhash** (uint blocknumber): o hash de um determinado bloco.
- **block.timestamp** (uint): tempo em segundos (unix) de um bloco, e muitas outras.

A lista completa de variáveis que podemos acessar, está disponível em: <https://solidity.readthedocs.io/en/v0.5.1/units-and-global-variables.html>.

### 5.3.6 Estudo de Caso: Conta Bancária

No exemplo abaixo, criamos a simulação de uma conta bancária, explorando os conceitos apresentados anteriormente.

```

1 pragma solidity ^0.5.1;
2
3 // declararemos uma interface para o contrato Banco
4 interface InterfaceBanco {
5     function verificaValor(uint valor) external returns (bool);
6 }
7
8 // Banco expande os métodos da interface
9 contract Banco is InterfaceBanco {
10    uint private saldo;
11    address private dono;
12
13    // Declaramos um modifier para verificar se o dono da conta
14    // eh quem esta executando as funções
15    modifier donoFunc {
16        require(dono == msg.sender);
17        -
18    }
19
20    constructor(uint valor) public {
21        saldo = valor;
22        dono = msg.sender;
23    }
24
25    function depósito(uint valor) public donoFunc {
26        saldo += valor;
27    }
28
29    function saque(uint valor) public donoFunc {
30        if (verificaValor(valor)) {
31            saldo -= valor;
32        }
33    }
34
35    function verificaSaldo() public view returns (uint) {
36        return saldo;
37    }
38
39    function verificaValor(uint valor) public returns (bool) {
40        return saldo >= valor;

```

```

41     }
42 }
43 // MeuContrato expande o contrato Banco (heranca)
44 // Inicialmente esta conta terá 20 ethers de saldo
45 contract MeuContrato is Banco(20 ether) {
46     string private nome;
47     uint private idade;
48
49     constructor(string memory entrada_nome, uint entrada_idade) public{
50         nome = entrada_nome;
51         idade = entrada_idade;
52     }
53
54     function getNome() public view returns (string memory) {
55         return nome;
56     }
57
58     function getIdade() public view returns (uint) {
59         return idade;
60     }
61 }
```

Alguns elementos do exemplo que podemos pontuar:

#### **Interface: InterfaceBanco**

Interfaces de contratos são declaradas com a palavra **interface**. Elas são utilizadas para especificar os métodos de um contrato, mas sem implementá-los. Não possuem construtor, não podem especificar variáveis de estado e suas funções são declaradas como **external**.

**Observação 5.3.2** Interfaces são semelhantes à classes abstratas, um conceito da orientação a objetos.

#### **Contrato: Banco**

O contrato **Banco** implementa os métodos da sua interface e **não é instanciado, pois serve apenas como modelo do contrato MeuContrato**. Assim, possui:

- Atributo **saldo**, que armazena seu saldo em Ether (unsigned int).
- Atributo **dono** que armazena o **endereço** do dono da “conta bancária”.
- Construtor que recebe o valor do saldo inicial da conta, indicado na instanciação do contrato.
- Funções **depósito** e **saque** que implementam as funções básicas de uma conta bancária. Utilizam da função modifier para verificação.
- Função **verificaSaldo** que retorna o saldo disponível da conta.
- Função **verificaValor** utilizada para verificar se o valor que se deseja sacar é menor que o saldo disponível.
- Função modifier **donoFunc** utilizada para verificar se a pessoa realizando as operações de saque e depósito, é o próprio dono da conta.

#### **Contrato: MeuContrato**

É o contrato que herda os métodos do contrato Banco. Nele setamos um saldo inicial de **20 ether** para toda conta. Além disso, possui:

- Atributos **nome** e **idade** para receber informações pessoais do dono da conta, que são definidas pelo construtor.
- Funções **getter** simples que retornam as informações do dono da conta.

#### **5.3.7 Vamos Exercitar?**

**Exercício 5.1** Compile, faça o deploy e instancie o contrato **MeuContrato** apresentado anteriormente. Em seguida, adicione novos atributos aos contratos, como: **CPF**, **Endereço**, **ID de Conta** e crie novas funções que julgue essenciais em uma conta bancária. Por fim, altere seu contrato de modo que uma conta aceite depósitos de outros usuários além do próprio dono. ■

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### 6. Estudo Guiado

#### 6.1 Ethereum Improvement Proposals (EIP)

Segundo a documentação oficial, EIPs são descrições de padrões para a plataforma Ethereum, incluindo especificações de protocolos, APIs de clientes, padrões de contratos, etc. Nessa apostila trabalharemos com os **Ethereum Request for Comment (ERC)**, um tipo de EIP onde relatam-se padrões em nível de aplicação. Isso significa que as aplicações mais comuns de contratos inteligentes já são documentadas de uma maneira **recomendada** pela comunidade, de forma a evitar possíveis problemas na rede e na própria aplicação.

**Observação 6.1.1** Para entender melhor a importância das ERC no contexto de aplicações descentralizadas, recomendamos a leitura do artigo “**A survey of attacks on Ethereum smart contracts**” que relata vulnerabilidades em contratos inteligentes, incluindo o caso conhecido como **DAO attack**, de 2016, onde aproximadamente U\$ 70 milhões de dólares foram “roubados”.

#### 6.2 Criptomoeda e Token

Na comunidade, a diferença entre **Criptomoeda** e **Token** não é muito bem consensual e a definição não é clara. Portanto, por convenção, nessa apostila adotamos o termo **criptomoeda** para definir as moedas digitais que estão inseridas no *núcleo* de uma Blockchain, como o Bitcoin e o Ether. E o termo **token**, usamos para indicar também moedas digitais, porém, moedas que são implementadas “*paralelamente*” a uma criptomoeda, fazendo uso de uma Blockchain que já possui sua moeda oficial.

No capítulo anterior, vimos que é possível criar uma infinidade de aplicações em Ethereum, utilizando da linguagem Solidity. Aproveitando-se desse conceito, empresas e programadores já criaram suas próprias moedas digitais dentro da rede Ethereum, como exemplo podemos citar: **Binance Coin**, **Tether USD** e o **TrueUSD**. Essas e aproximadamente outras **200 mil** moedas possuem algo em comum: foram criadas utilizando do **ERC-20**, que é um padrão para implementação de **tokens** digitais.

**Observação 6.2.1** Uma lista completa dos tokens implementadas no Ethereum, pode ser acessada pelo link: <https://etherscan.io/tokens>. Porém, nem todas possuem um valor de mercado significativo.

## 6.3 Criando nosso Token

Nessa seção, faremos a implementação prática de um token utilizando a linguagem Solidity. Para isso, usaremos dois padrões de implementação, o ERC-20 e o ERC-721. Mas, para entender a diferença entre eles, primeiramente precisamos a entender o conceito de fungibilidade.

Um bem fungível, é um bem que pode ser trocado por outro de mesma espécie, qualidade e quantidade sem perder o seu valor absoluto, como, por exemplo, uma nota de \$50 reais. Se irmos a uma loja, e trocarmos nossa nota de \$50 por outra nota de \$50, ambas terão o mesmo valor para a sociedade, desde que ambas tenham sido emitidas pelo Banco Central.

Já um bem infungível (ou não-fungível) é o oposto, por exemplo, uma obra de arte ou itens colecionáveis. Se trocarmos a pintura original da **Monalisa** por outra Monalisa, a original terá um valor de mercado muito maior que a não-original. E, mesmo que **ambas** tenham sido pintadas pelo próprio Leonardo Da Vinci, ainda assim, é bem provável que não terão o mesmo valor significativo. Dessa forma, foi definido:

- **ERC-20**: Padrão de implementação de tokens fungíveis.
- **ERC-721**: Padrão de implementação de tokens não-fungíveis.

Além desses, ainda há outros padrões que exploram a criação de tokens (por exemplo, o ERC-223) mas não chegaram a ser tão “aceitos” pela comunidade como esses dois.

### 6.3.1 ERC-20

O ERC-20 define um conjunto de seis métodos e dois eventos para transferência de ativos digitais. No exemplo a seguir, apresentamos uma **possível** implementação dessas funções, mas tenha em mente que existem inúmeras implementações diferentes que são adaptadas ao contexto das suas respectivas aplicações, mas todas seguem o mesmo “fluxo” de representação.

Seguimos os padrões de nomenclatura que são adotados pela comunidade, assim o leitor estará mais familiarizado com outras implementações. Dessa forma:

- A nomenclatura dos métodos e variáveis estão em **inglês**.
- Atributos do contrato **MeuToken** que são utilizados para controle de fluxo nas funções, são precedidos com “`_`”.
- Variáveis passadas como parâmetros das funções são precedidas com “`_`”.

```

1 pragma solidity ^0.5.1;
2
3 interface ERC20 {
4
5     // Definindo as funções
6     function totalSupply() external view returns (uint _totalSupply);
7     function balanceOf(address _owner) external view returns (uint balance);
8     function transfer(address _to, uint _value) external returns (bool success);
9     function transferFrom(address _from, address _to, uint _value) external
10        returns (bool success);
11     function approve(address _spender, uint _value) external
12        returns (bool success);
13     function allowance(address _owner, address _spender) external view
14        returns (uint remaining);
15
16     // Definindo os eventos
17     event Transfer(address indexed _from, address indexed _to, uint _value);
18     event Approval(address indexed _owner, address indexed _spender, uint _value);
19 }
20
21 contract MeuToken is ERC20 {
22
23     string public name;
24     string public symbol;
25     uint8 public decimals;
26     uint private __totalSupply;
27     mapping (address => uint) private __balanceOf;
28     mapping (address => mapping (address => uint)) private __allowances;
29
30     constructor() public {
31         name = "Meu Primeiro Token";
32         symbol = "MPT";
33         decimals = 18;

```

```

34     __totalSupply = 1000;
35     __balanceOf[msg.sender] = __totalSupply;
36 }
37
38 function totalSupply() public view returns (uint _totalSupply) {
39     _totalSupply = __totalSupply;
40 }
41
42 function balanceOf(address _addr) public view returns (uint balance) {
43     return __balanceOf[_addr];
44 }
45
46 function transfer(address _to, uint _value) public returns (bool success) {
47     if (_value > 0 && _value <= balanceOf(msg.sender)) {
48         __balanceOf[msg.sender] -= _value;
49         __balanceOf[_to] += _value;
50         return true;
51         emit Transfer(msg.sender, _to, _value);
52     }
53     return false;
54 }
55
56 function transferFrom(address _from, address _to, uint _value) public
57 returns (bool success) {
58     if (__allowances[_from][msg.sender] > 0 &&
59         _value > 0 &&
60         __allowances[_from][msg.sender] >= _value &&
61         __balanceOf[_from] >= _value) {
62         __balanceOf[_from] -= _value;
63         __balanceOf[_to] += _value;
64         __allowances[_from][msg.sender] -= _value;
65         emit Transfer(_from, _to, _value);
66         return true;
67     }
68     return false;
69 }
70
71 function approve(address _spender, uint _value) public returns (bool success) {
72     __allowances[msg.sender][_spender] = _value;
73     emit Approval(msg.sender, _spender, _value);
74     return true;
75 }
76
77 function allowance(address _owner, address _spender) public view
78 returns (uint remaining) {
79     return __allowances[_owner][_spender];
80 }
81 }
```

**Exercício 6.1** Compile, instancie e insira casos de teste no contrato **MeuToken** apresentado anteriormente. Não se esqueça de trabalhar com as inúmeras “**Accounts**” disponibilizadas pelo Remix, fazendo transferências e emitindo transações para esse contrato.

### Interface do Token

Na interface **ERC20** definimos as seis funções essenciais de um token ERC-20: **totalSupply**, **balanceOf**, **transfer**, **transferFrom**, **approve** e **allowance**. E, também, dois eventos que são utilizados para registrar o log das transações na Blockchain: **Transfer** e **Approval**. Lembrando que toda função declarada em uma interface, deve ser definida como **external**, e aquelas que apenas retornam um valor sem alterá-lo, são definidas como **view**.

**Observação 6.3.1 — Comentário.** Não é necessário indicar o nome da uma variável de retorno. Por exemplo, na linha 6 poderíamos fazer: **function totalSupply() external view returns (uint)**. Porém, lembre-se de fazer as mudanças da função também no contrato **MeuToken**, removendo a variável auxiliar.

### Atributos do Token ERC-20

No contrato **MeuToken**, definimos alguns atributos usados para caracterizar nosso token, como:

- **name**: onde definimos o nome do nosso token.
- **symbol**: utilizado para referencia, como o ETH é para o Ether.
- **decimals**: número de casa decimais da unidade atômica do nosso token, como o centavo é para o real. Por padrão, é definido 18, mas fica a critério do programador.
- **totalSupply**: quantidade **total** de unidades existentes do nosso token.
- **\_balanceOf**: estrutura de dados global semelhante a uma **Hash Table**. Ela irá armazenar a quantidade que cada pessoa (address) possui do nosso token.
- **\_allowances**: estrutura que receberá a quantidade que uma pessoa (address) permite a outra sacar da sua conta. Em sequência entenderemos melhor esse conceito.

### Consulta de Saldo e Aporte Total

Para sabermos o saldo que uma pessoa possui do nosso token, basta executarmos a função **balanceOf (address)**, onde o parâmetro **address** é o endereço dessa pessoa. E, para verificarmos a quantidade (ou,aporte) total de tokens existentes, executamos a função **totalSupply()**. Tenha em mente que nesse exemplo, o aporte total não poderá ser modificado posteriormente, portanto, não será possível fazer a “emissão” de novos tokens após o *Deploy* do contrato.

### Tipos de Transferências: *Transfer* e *TransferFrom*

No padrão ERC-20, foram definidos duas maneiras distintas de transferir tokens:

1. **Transferência direta**: definida pela função **transfer**. Basta a pessoa inserir o endereço de destino, a quantidade de tokens que será transferida, e executar a transação. Caso os requisitos necessários sejam atendidos, o endereço destino irá receber a quantia.
2. **Transferência indireta**: é uma maneira alternativa de fazer transferências. Inicialmente, a pessoa irá emitir uma **permissão** de transferência para outra pessoa através da função **approve (spender, value)**, onde o **spender** é aquele que será permissionado e o **value** é o valor permitido. Em seguida, a pessoa permissionada a fazer a transferência executa a função **transferFrom (from, to, value)**, indicando o endereço de origem **from** (aquele que permitiu a transferência), o endereço de destino **to** (quem irá receber aquele token) e a quantia transferida.

Executando a função **allowance (owner, spender)** podemos verificar a quantia que uma pessoa **owner** permitiu a pessoa **spender** transferir da sua conta. Para isso, é utilizada a estrutura **\_allowances** citada anteriormente.

### Eventos Gerados

Eventos são utilizados para fazer o **log** de uma transação executada, onde são salvos no *transactions log*, uma estrutura especial do Ethereum. Em geral, são utilizados para facilitar a comunicação com o front-end de DApps via protocolo web3. Para isso, sempre que as condições de transferência ou aprovação de uma transferência seja completos, acionamos um evento com a palavra reservada **emit**.

### Vamos Exercitar?

**Exercício 6.2** Talvez você tenha notado que no nosso token ERC-20 utilizamos a estrutura **if** para as condições de transferências. Porém, como já dizemos anteriormente, o **if** é muito custoso em Solidity e não é muito recomendado. Portanto, adapte o código anterior utilizando das estruturas **require** e/ou **assert**, substituindo os **if's**. ▀

**Exercício 6.3** No código anterior, os tokens são transferidos apenas em unidades inteiras (**uint**). Utilizando das unidades de Ether (**wei**, **szabo**, **finney** e **ether**), altere seu código para que as transações possam ser feitas em unidades decimais atômicas. Detalhe: até o momento de escrita dessa apostila, o Solidity não suporta variáveis do tipo **float** ou **double** como outras linguagens, use a criatividade. ▀

Mais detalhes e informações sobre o ERC-20 podem ser encontradas na documentação oficial, disponível em: <https://eips.ethereum.org/EIPS/eip-20>.

### 6.3.2 ERC-721

Com o ERC-721 faremos uma abordagem diferente. Apresentaremos ao leitor, a interface com os métodos e atributos do padrão, seguido por uma explicação do seu funcionamento. Em seguida, o leitor deverá fazer a própria implementação. Também apresentaremos materiais complementares que podem auxiliá-lo no desenvolvimento.

O ERC-721 foi escrito de modo a ser familiar para o usuário acostumado com o ERC-20. Assim, grande partes das funções e dos eventos são implementados seguindo o mesmo padrão, apenas adaptados a um contexto de infungibilidade.

```

1 pragma solidity ^0.5.1;
2
3 interface ERC721 {
4
5     // Funções semelhantes ao ERC-20
6     function totalSupply() pure external returns (uint256 totalSupply);
7     function balanceOf(address _owner) pure external returns (uint balance);
8
9     // Funções específicas do ERC-721
10    function ownerOf(uint256 _tokenId) pure external returns (address owner);
11    function approve(address _to, uint256 _tokenId) external;
12    function takeOwnership(uint256 _tokenId) external;
13    function transfer(address _to, uint256 _tokenId) external;
14    function tokenOfOwnerByIndex(address _owner, uint256 _index) pure external
15        returns (uint tokenId);
16
17    // Metadados do Token
18    function tokenMetadata(uint256 _tokenId) pure external returns (string memory
19        infoUrl);
20
21    // Eventos semelhantes ao ERC-20
22    event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
23    event Approval(address indexed _owner, address indexed _approved, uint256
24        _tokenId);
25 }
```

#### Atributos do Token ERC-721

Na interface **ERC721**, especificamos alguns métodos que podem ser descritos da seguinte forma:

- **OwnerOf**: retorna o **endereço** do dono de um token específico.
- **Approve**: permite que outra entidade faça a transferência do seu token em um momento posterior (transferência indireta).
- **takeOwnership**: função usada para “sacar” um token que o usuário foi autorizado a sacar.
- **transfer**: função usada para fazer a transferência direta da propriedade de um token de um endereço para outro.
- **tokenOfOwnerByIndex**: array utilizado para armazenar todos os tokens que um usuário possui. Isso reduz a complexidade de fazer a busca de tokens de um usuário nas estruturas globais.
- **tokenMetadata**: função responsável por indicar as propriedades intrínsecas de cada token. Por exemplo, um carro possui suas características únicas como: a placa, o número de identificação do veículo (chassi), além das suas características físicas e outros atributos.

Boas referências para implementações podem ser encontradas em: <http://erc721.org/>, <https://medium.com/crypto-currently/the-anatomy-of-erc721-e9db77abfc24> e <https://blockgeeks.com/guides/erc-721-cryptokitty-token/>.

**Exercício 6.4** Faça as implementações necessárias das funções, de modo que seu contrato possua uma quantidade limitada de cinco modelos de carros populares. Por exemplo, 120 carros do modelo **Pálio**, 30 do modelo **Gol**, etc. E, permita que os usuários (endereços) possam fazer a troca desses carros.

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### 7. Ferramentas Utilizadas



#### 7.1 Ferramentas e Notícias

##### 7.1.1 Diagramas

Draw.io sob licença **Free-to-license**.

##### 7.1.2 Ícones

Fontawesome, IBM Blockchain Icon e Flaticon sob licenças **Creative Commons License**.

##### 7.1.3 Captura de Código

Carbon, disponível em: <https://carbon.now.sh/>

##### 7.1.4 LaTeX

###### Template

“The Legrand Orange Book” de Mathias Legrand (legrand.mathias@gmail.com) com modificações de Lucas (lucasribeiro@furg.br) sob licença CC BY-NC-SA 3.0, disponível em: <http://creativecommons.org/licenses/by-nc-sa/3.0/>

###### Tag

Solidity LaTeX Highlighting.

#### 7.2 Notícias

1. “Teresina usará tecnologia Blockchain no Transporte Público”. Publicado em [clp.org](http://clp.org)
2. “Carrefour and Nestlé Partner with IBM to Extend Use of Blockchain to New Food Categories”. Publicado em: [ibm.com](http://ibm.com)
3. “8 países se adequando a tecnologia blockchain”. Publicado em: [livecoins.com.br](http://livecoins.com.br)
4. “Teste feito por equipe da Unicamp revelou falhas de segurança nas urnas eletrônicas”. Publicado em: [senado.leg.br](http://senado.leg.br)
5. “The difference between Token and Cryptocurrency”. Publicado em: [medium.com](http://medium.com) por Marco Cavicchioli.
6. “Explainer: what is blockchain, who's using it and why?”. Publicado em: [unilever.com](http://unilever.com)
7. “Votação eletrônica é realidade em mais de 30 países”. Publicado em: [tse.jus.br](http://tse.jus.br)

8. “Transactions in Ethereum”. Publicado em: [medium.com](#) por KC Tam.

**Observação 7.2.1 — Aviso.** Esquecemos de alguma referência ou crédito? Por favor, entre em contato por um dos e-mails: lucasribeiro@furg.br ou odorico.mendizabal@ufsc.br com o assunto “Apostila Blockchain”.

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### Bibliografia

#### Artigos

- [1] Merlinda Andoni et al. “Blockchain technology in the energy sector: A systematic review of challenges and opportunities”. Em: *Renewable and Sustainable Energy Reviews* 100 (2019), páginas 143-174 (ver página 17).
- [3] Vitalik Buterin et al. “Ethereum white paper”. Em: *Github repository* (2013), páginas 22-23 (ver página 33).
- [4] Christian Cachin e Marko Vukoli. “Blockchain consensus protocols in the wild”. Em: *arXiv preprint arXiv:1707.01873* (2017) (ver página 25).
- [7] Yuliya Demanyk e Otto Van Hemert. “Understanding the subprime mortgage crisis”. Em: *The Review of Financial Studies* 24.6 (2009), páginas 1848-1880 (ver página 10).
- [9] Johannes Göbel et al. “Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay”. Em: *Performance Evaluation* 104 (2016), páginas 23-41 (ver página 24).
- [11] Sunny King e Scott Nadal. “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake”. Em: *self-published paper, August 19* (2012) (ver página 25).
- [13] Leslie Lamport. “Time, clocks, and the ordering of events in a distributed system”. Em: *Communications of the ACM* 21.7 (1978), páginas 558-565 (ver página 26).
- [14] Leslie Lamport, Robert Shostak e Marshall Pease. “The Byzantine generals problem”. Em: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), páginas 382-401 (ver página 26).
- [15] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. Em: (2008). url: <http://bitcoin.org/bitcoin.pdf> (ver páginas 10, 22, 25).
- [16] Arvind Narayanan e Jeremy Clark. “Bitcoin’s academic pedigree”. Em: *Communications of the ACM* 60.12 (2017), páginas 36-45 (ver página 26).
- [18] Kelly Olson et al. “Sawtooth: An Introduction”. Em: *The Linux Foundation, Jan* (2018) (ver página 26).
- [19] Carlos Perez. “EVM and Assembly”. Em: *Github Repository* (2018) (ver página 29).
- [20] Marc Pilkington. “Il Blockchain technology: principles and applications”. Em: *Research handbook on digital transformations* 225 (2016) (ver página 21).

- [22] Ken Shirriff. "Mining Bitcoin with Pencil and Paper: 0.67 Hashes per Day". Em: *Ken Shirriff's Blog* (2014) (ver página 23).
- [24] Nick Szabo. "Smart contracts". Em: *Unpublished manuscript* (1994). url: <http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html> (ver página 26).
- [25] John Turek e Dennis Shasha. "The many faces of consensus in distributed systems". Em: *Computer* 25.6 (1992), páginas 8-17 (ver página 24).
- [28] Wikipédia. "Sistema de reserva fracionária — Wikipédia, a enclopédia livre". Em: (2018). [Online; accessed 30-maio-2018]. url: [https://pt.wikipedia.org/w/index.php?title=Sistema\\_de\\_reserva\\_fracion%C3%A1ria&oldid=52225107](https://pt.wikipedia.org/w/index.php?title=Sistema_de_reserva_fracion%C3%A1ria&oldid=52225107) (ver página 9).
- [29] Gavin Wood. "Ethereum yellow paper". Em: *Internet: https://github.com/ethereum/yellowpaper/[Oct. 30, 2018]* (2014) (ver página 30).

## Livros

- [2] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. "O'Reilly Media, Inc.", 2014 (ver páginas 22, 24).
- [17] Arvind Narayanan et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016 (ver página 22).
- [23] William Stallings, Graça Bressan e Akio Barbosa. *Criptografia e segurança de redes*. Pearson Educación, 2008 (ver página 23).

## Outros

- [5] Miguel Castro, Barbara Liskov et al. "Practical Byzantine fault tolerance". Em: *OSDI*. Volume 99. 1999. 1999, páginas 173-186 (ver páginas 26, 27).
- [6] Nutthakorn Chalaemwongwan e Werasak Kurutach. "State of the art and challenges facing consensus protocols on blockchain". Em: *2018 International Conference on Information Networking (ICOIN)*. IEEE. 2018, páginas 957-962 (ver página 25).
- [8] Cynthia Dwork e Moni Naor. "Pricing via processing or combatting junk mail". Em: *Annual International Cryptology Conference*. Springer. 1992, páginas 139-147 (ver página 26).
- [10] Stuart Haber e W Scott Stornetta. "How to time-stamp a digital document". Em: *Conference on the Theory and Application of Cryptography*. Springer. 1990, páginas 437-455 (ver página 10).
- [12] Joshua A Kroll, Ian C Davey e Edward W Felten. "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries". Em: *Proceedings of WEIS*. Volume 2013. 2013 (ver página 25).
- [21] Lakshmi Siva Sankar, M Sindhu e M Sethumadhavan. "Survey of consensus protocols on blockchain applications". Em: *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE. 2017, páginas 1-5 (ver página 25).
- [26] Marko Vukoli. "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication". Em: *International workshop on open problems in network security*. Springer. 2015, páginas 112-125 (ver página 26).
- [27] Marko Vukoli. "Rethinking permissioned blockchains". Em: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, páginas 3-7 (ver página 27).

# APOSTILA PARA INICIANTES

## INTRODUÇÃO À BLOCKCHAIN E CONTRATOS INTELIGENTES

2019

### Índice

B
---

Blockchain permissionada .....	22
--------------------------------	----

Bloco .....	22
-------------	----

### C

Cadeia de blocos.....	21
-----------------------	----

Carteira .....	23
----------------	----

Consenso.....	24
---------------	----

Contexto .....	8
----------------	---

Contratos Inteligentes.....	36
-----------------------------	----

Criando nosso Token .....	47
---------------------------	----

Criando seu Token
-------------------

ERC-20 .....	47
--------------	----

ERC-721 .....	50
---------------	----

Criptografia .....	23
--------------------	----

### D

DApps .....	33
-------------	----

### E

Especificações .....	28
----------------------	----

Consenso Ethereum.....	33
------------------------	----

Documentação .....	33
--------------------	----

Evolução.....	9
---------------	---

### F

Ferramentas Utilizadas .....	51
------------------------------	----

### G

Governo e Sociedade .....	18
---------------------------	----

Dados Públicos .....	18
----------------------	----

Gestão de Saúde .....	19
-----------------------	----

Transporte Urbano .....	19
-------------------------	----

Votações .....	18
----------------	----

### H

Hyperledger Fabric .....	27
--------------------------	----

### I

IBM Hyperledger .....	27
-----------------------	----

Indústrias .....	17
------------------	----

Energética .....	17
------------------	----

Farmacêutica e Alimentícia.....	18
---------------------------------	----

### M

Mercado Financeiro.....	14
-------------------------	----

Mineração .....	24
-----------------	----

### N

Nodos completos .....	24
-----------------------	----

### O

Objetivos .....	11
-----------------	----

**P**

Plataformas.....	27
EOS .....	27
Ethereum .....	27
POeT, <i>Proof of elapsed time</i> .....	26
PoS, Prova de participação .....	25
PoW, Prova de trabalho.....	25
Practical Byzantine Fault Tolerance, PBFT..	26
Proposta.....	10
Prova de tempo decorrido.....	26

**R**

Remix .....	36
-------------	----

**T**

Tabelionato e Registros.....	16
------------------------------	----